On Real Time Digital Phase Locked Loop Implementation with Application to Timing Recovery

Roger Kippenberger, B.E.(hons)

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering
in
Electrical and Electronic Engineering
at the
University of Canterbury,
Christchurch, New Zealand

Communications Research Group
Department of Electrical and Electronic Engineering
University of Canterbury,
Christchurch, New Zealand
November 2006

Abstract

In digital communication systems symbol timing recovery is of fundamental importance. The accuracy in estimation of symbol timing has a direct effect on received data error rates. The primary objective of this thesis is to implement a practical Digital Phase Locked Loop capable of accurate synchronisation of symbols suffering channel corruption typical of modern mobile communications.

This thesis describes an all-software implementation of a Digital Phase Locked in a real-time system. A timing error detection (TED) algorithm is optimally implemented into a Digital Signal Processor. A real-time transmitter and receiver system is implemented in order to measure performance when the received signal is corrupted by both Additive White Gaussian Noise and Flat Fading.

The Timing Error Detection algorithm implemented is a discrete time maximum likelihood one known as FFML1, developed at Canterbury University. FFML1 along with other components of the Digital Phase Locked loop are implemented entirely in software, using Motorola 56321 assembly language.

Acknowledgments

Firstly, I would like to thank my supervisor Professor Des Taylor, whose patience must have been sorely tested as I took the scenic route to completion of this thesis. His generosity and advice are were very much appreciated.

The support of my wife, Kerri, was as it always is, unwaivering. For that support I owe the opportunity to have undertaken this work. Our daughter, Lucinda born ten weeks premature right when I had set aside for completion of this writing was just one of the hurdles to overcome.

Finally I'd like to thank my extended family who all at some point have provided support and assistance that has enable me to undertake my postgraduate ambitions. Mum, Dad, Jeannette, Garry, Dean and Kay, cheers.

Roger Kippenberger,

November 2006.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Software Defined radio

In todays commercial environment the radio frequency spectrum is a resource under ever increasing demand. With services already established throughout the usable radio spectrum regulatory agencies are increasingly looking to *sharing* mechanisms and technologies that apply higher degrees of efficiency to the use of the radio spectrum resource. A 2003 FCC report on *The Future of Spectrum Policy* [1] states

> The Task Force's own findings support the conclusion that whereas the analog era may have justified a government grant of exclusive rights to a band of frequencies, the development of digital and software-defined ("smart") radio technologies will make it feasible for individual citizens to dynamically share wide ranges of underutilized spectrum without imposing harmful interference on licensed or on other unlicensed users.

Ideally a Software Defined Radio (SDR) (or smart radio) transposes as much as possible of the signal processing from traditional analog methods to software running on a Digital Signal Processor (DSP). Typically the only analog processing required required on an SDR front end is a super-heterodyne is that translates the radio frequency signal between a carrier frequency and some analog intermediate frequency (IF) . Conversion between the analog IF signal and the digital domain is then achieved with the use of either an Analog to Digital (ADC) and bandpass sampling process or Digital to Analog (DAC)

conversion. In some cases the analog signal processing employed in a traditional radio is simply transposed to its equivalent in the digital domain, but in other cases digital processing allows new methodologies, offering increased efficiency and performance of the SDR compared with its analog counterpart.

The most significant advantage of SDR's over traditional analog radios is that being defined in software enables a radio to dynamically change and reconfigure operating parameters such as modulation, transmission rate and frequency [2]. Dynamic reassignment of these operational quantities coupled with *spectrum sensing*[1] technology enables a radio to use spectrum adaptively, to select frequency and bandwidth usage *on the fly* for efficient spectral usage and to radiate only as much power as required for reliable communication. This dynamic access and usage of spectrum is known as *cognitive radio* [3] and is an area of active research and of great interest to regulatory bodies seeking to increase efficiency and available capacity of the radio frequency spectrum.

1.2 Scope of Thesis

This thesis seeks to develop one aspect of SDR, the implementation of a Digital Phase Locked Loop (DPLL) for symbol clock recovery in software. To analyse performance and demonstrate capabilities this DPLL will be integrated into an operational digital transmission system employing SDR principals.

The DPLL will employ a Timing Error Detection algorithm developed at Canterbury University called FFML1 [4], an algorithm that has demonstrated some promise in its ability to acquire and track the phase of a digitally modulated signal in an fading environment using with a DSP based software implementation. FFML1 was designed specifically with the flat fading channel in mind, but this ability comes at a cost of significantly increased processing demand over traditional methods. So one of the key goals of this thesis is to implement an FFML1 based DPLL in an optimal way and demonstrate that it can operate in a practical environment at useful data rates. Various implementations that would inherently reduce the execution time of the DPLL loop were investigated. These included reducing the sampling rate per symbol, reducing the complexity of the symbol constel-

[1] *Spectrum Sensing* is the ability to measure spectrum usage over a wide bandwidth.

lation to single level modulation scheme and using a reduced complexity interpolation method.

Firstly the DPLL will be encoded into DSP using low-level assembly language for maximum efficiency. This implementation will be tested using simulated received symbol samples corrupted by varying levels of both Additive White Gaussian Noise (AWGN) and flat fading. Once the software design of the DPLL is proven, a transmitter receiver system will be constructed with the DPLL interfaced to the receiver. The objective of this system will be to demonstrate both the successful reception of digital information and the operation of the DPLL in a real-time system.

1.3 Thesis Organisation

This subsection briefly describes the following chapters of this thesis.

Chapter two provides background for the use of mathematical representation of signals, modulation and symbols. It also describes methodology for simulation of AWGN and flat fading channel corruption. Other relevant fundamentals of digital communications are also presented.

Chapter three introduces the concepts of receiver timing synchronisation and the relationship between the accuracy of symbol timing recovery and receiver error rates. The traditional Phase Locked Loop (PLL) is introduced along with associated analog processing techniques required to produce symbol estimates. Digital techniques for achieving the same symbol estimate are presented, showing how the building blocks of the PLL can be transposed into the digital domain, resulting in the DPLL. Finally a generic model for a DPLL is presented, forming the basis for the software design presented in the succeeding chapter.

Chapter four describes the DPLL software design that is implemented into DSP. The design is broken down into functional blocks and the design of each block is discussed. This software design is then tested using MatLab scripts, and the theoretical performance presented. The software design is then ported to DSP assembly language and the operation tested using simulated received symbol samples.

Chapter five continues the development by integrating the DPLL design into a real-

time transmitter receiver system, SASRATS [5]. Performance of the DPLL is measured as the receiver operates in a practical implementation. This demonstrates the basic viability of the use of the DPLL.

Chapter six summarises previous chapters presenting conclusions and suggesting directions for future research.

Chapter 2

Digital Communications Background

2.1 Introduction

Digital communication is a means of transmitting discrete, digital information through what is essentially an analog medium. This chapter provides some basic theory of digital *modulation* and then describes some relevant mathematical tools and channel statistics and models that will be employed in later chapters.

The concept of a *vector channel* [6] is introduced as it is especially useful for the representation of digital signals. Concepts of vector mathematics can be conveniently applied to discrete samples of a digitally modulated signal, where a complex signal sample can be considered as a two-dimensional vector. A basic digital modulation scheme and one that is employed throughout this thesis is that of quarternary phase shift keying (QPSK). A simple system by which a QPSK signal can be constructed is provided to give the reader a conceptual basis for QPSK modulation. A *differential* coding scheme is shown that can be applied to QPSK. Also described is the recovery of data at the receiver, which depends on the estimation of transmitted vectors, leading to the development of vector based *decision regions*.

In reality the transmission medium or channel through which a signal must pass will subject the signal to distortion and usually spectral limitation. Typical characteristics encountered in a mobile channel are described; how path loss is modeled, a description of a fading channel, how a fading channel is formed and how that fading can be modeled. As the radio spectrum is shared with other users, it is also typical for radio based com-

munications systems to be required to limit frequencies radiated through the air by the transmitter to be within some defined channel. So *pulse shaping* is introduced as a means of limiting the bandwidth of transmission.

2.2 Signal Representation, Modulation and Coding

This section describes the representation of data being transmitted as messages or *signal vectors* that can be modulated onto a passband signal at some carrier frequency.

2.2.1 Bandpass Transmission for Digital Communications

A representation of a digital communication system is described in [7] and is modeled in terms of a series of components as shown in figure 2.1.

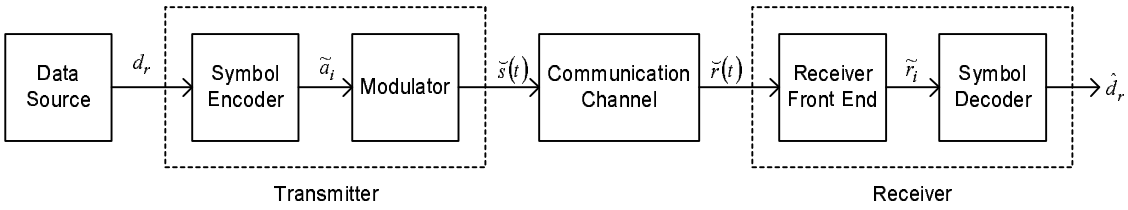

Figure 2.1: Digital Transmission Model

where the variables in the above diagram are defined explicitly throughout the course of the chapter. In brief the components of the system consist of the following:

- The *data source* d_r can be assumed to be a source of discrete, information bearing binary information, usually binary bits at some rate R bits/s
- The *symbol encoder* maps the data onto a vector space to form (complex) symbols $\tilde{a}_i$
- The *modulator* maps symbols onto a carrier, usually a sinusoid, for transmission as a bandpass signal $\breve{s}(t)$
- The *communication channel* is the medium through which the transmitted signal passes from one point to another, usually corrupting the signal with additive noise,

and possibly multiplicative distortion and/or inter symbol interference (ISI) producing a received signal $\check{r}(t)$

- The *receiver front end* estimates transmitted symbol vectors from the received signal $\tilde{r}_i$
- The *symbol decoder* then attempts to estimate the most likely transmitted data $\hat{d}_r$ from the estimated signal vector information $\tilde{r}_i$

2.2.2 Vector Channels

The *digital* signal processing of complex signal samples is of primary importance to the work of this thesis. In practice, a digitally modulated signal will typically contain information in both in-phase *(I)* and quadrature *(Q)* components centered at some carrier frequency. These components are realised by utilising two real-time *basis functions*, φ^I and φ^Q, defined by [6] as

$$\begin{aligned} \varphi^I(t) &= \sqrt{\frac{2}{T}} cos(2\pi f_c t) \\ \varphi^Q(t) &= \sqrt{\frac{2}{T}} sin(2\pi f_c t) \\ & 0 \leq t \leq T \end{aligned} \tag{2.1}$$

where φ^I, φ^Q are *orthonormal* such that

$$\begin{aligned} \int_0^T \varphi^I(t) \cdot \varphi^Q(t) dt &= 0 \\ \int_0^T \varphi^Q(t) \cdot \varphi^Q(t) dt &= 1 \\ \int_0^T \varphi^I(t) \cdot \varphi^I(t) dt &= 1 \end{aligned} \tag{2.2}$$

A bandpass signal is then defined as the continuous signal

$$\breve{s}(t) \quad = \quad \sum_{i=1}^{\infty} \left[a_i^I \varphi_i^I(t) + a_i^Q \varphi_i^Q(t) \right] \tag{2.3}$$

where the coefficients a^I and a^Q can be represented the as components of a complex number as

$$\begin{aligned} \tilde{a}_i \quad &= \quad a_i^I + j \cdot a_i^Q \\ &\qquad iT \leq t \leq (i+1)T \end{aligned} \tag{2.4}$$

and $\tilde{a}_i$ is the *symbol* to be transmitted in the *ith* symbol period.

2.2.3 QPSK Waveform Construction

QPSK is characterised by a signal constellation of four message points mapped onto a two dimensional signal space. Information bearing symbols $\tilde{a}_i$ are *encoded* from consecutive bit pairs (dibits) of the data to be transmitted d_r, where the subscript r denotes an index with a rate twice that of the symbol rate denoted by index i indicating that the data rate is half that of the symbol rate. The real component a_i^I and imaginary component a_i^Q are encoded such that

$$a_i^I, a_i^Q \quad \in \quad \{-1, 1\} \tag{2.5}$$

In the most common method of producing a QPSK passband waveform, the coordinates $\left\{ a_i^I, a_i^Q \right\}$ are multiplied with the basis functions $\left\{ \varphi_i^I(t), \varphi_i^Q(t) \right\}$respectively and then summed as in figure 2.2, where f_c is the carrier centre frequency.

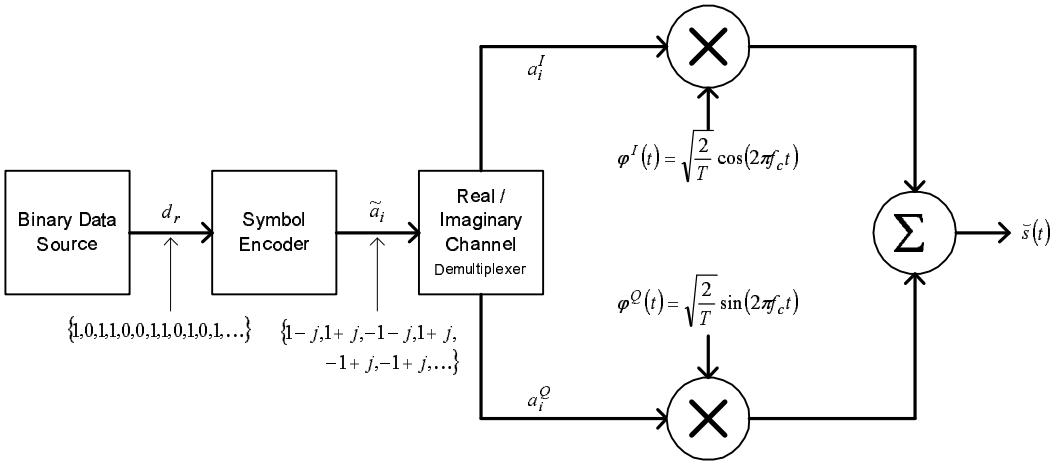

Figure 2.2: QPSK Transmitter Model, reproduced from [7] with some modification.

For a QPSK waveform, we assume that the symbol encoder maps dibits to coordinates as per table 2.1. This alphabet of possible symbols produces a symbol constellation that is nominally of *constant energy*, a characteristic that is exploited in later sections to simplify receiver design.

$\{d_r, d_{r+1}\}$	$\tilde{a}_i$
00	1+j
01	-1+j
11	-1-j
10	1-j

Table 2.1: QPSK Dibit to Symbol Mapping

The transmitter model of Figure 2.2 produces a bandpass signal such that the phase of signal $\breve{s}(t)$ is shifted between one of four points, representing each of the possible symbols. This passband signal $\breve{s}(t)$ may be written in terms of the symbol components a^I and a^Q as

$$\begin{aligned} \breve{s}(t) &= \sqrt{\frac{2}{T}}\left[a_i^I \cos(2\pi f_c t) + a_i^Q \sin(2\pi f_c t)\right] \\ & iT \leq t \leq (i+1)T \end{aligned} \tag{2.6}$$

2.2.4 Differential Coding

The QPSK modulation discussed in section 2.2.2 requires that absolute knowledge of the transmitted phase be recovered at the receiver in order for the correct symbol decision to be made. Alternatively coding can be applied to eliminate the need for recovery of the absolute phase; *differential coding* is a method where information is represented in terms of *transitions* instead of absolute position. In the case of QPSK, the phase difference between consecutive complex symbols can be used to represent the original data, resulting in a modulation known as differentially encoded QPSK (D-QPSK). For a symbol $\tilde{a}_i$ expressed in exponential form as

$$\tilde{a}_i = e^{k2\pi\theta_i} \tag{2.7}$$

the phase ϕ_i is determined by

$$\phi_i = \phi_{i-1} + \phi_d \tag{2.8}$$

where ϕ_d is mapped to the source binary data dibits as per table 2.2.

$\{d_r, d_{r+1}\}$	ϕ_d
00	0
01	$\frac{-\pi}{2}$
11	$\frac{\pi}{2}$
10	π

Table 2.2: Dibit to Phase Difference Mapping for D-QPSK Encoding

D-QPSK coding can be applied directly to the binary data sequence by replacing the QPSK encoder block of figure 2.2 with a differential encoder employing the above scheme. A transmitter employing D-QPSK modulation behaves exactly the same as one employing the QPSK modulation technique described in section 2.2.2. Recovery of the

data d_i then depends on the correct decoding of consecutive symbols in order to determine the transmitted phase ϕ_i.

2.2.5 Baseband Representation

It is often convenient to represent to represent a real bandpass signal as a complex baseband signal. For example, [8] employs the *down-mixed, low-pass* model of figure 2.3 to represent a baseband signal $\tilde{s}(t)$ generated from a bandpass QPSK signal $\check{s}(t)$.

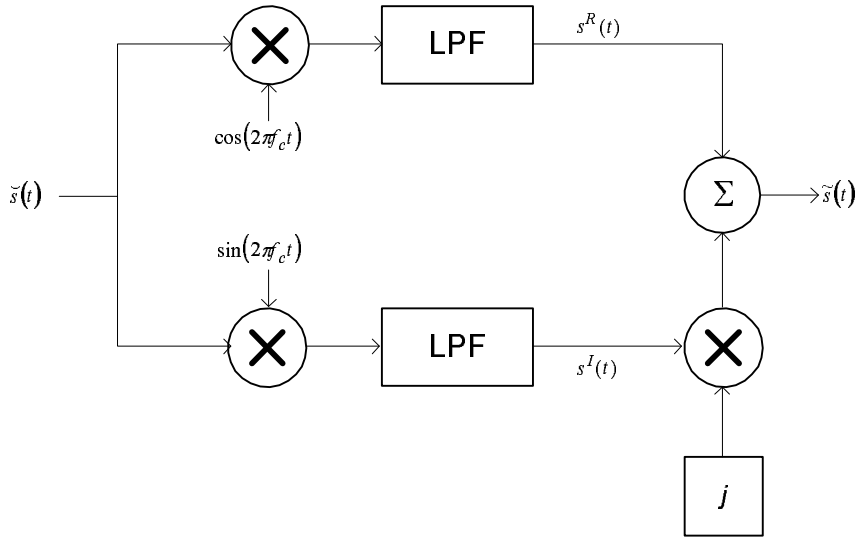

Figure 2.3: Method For Constructing the Low-Pass Representation From a Bandpass Signal reproduced from [8]

It is assumed that this system is *coherent*, in that both the frequency and phase of the carrier f_c are known. The *LPF* blocks imply low-pass filtering that removes the image frequency produced by the mixer.

We define the baseband signal $\tilde{s}(t)$ generated by the model of figure 2.3 as

$$\tilde{s}(t) = \frac{1}{\sqrt{T}} \cdot \sum_{i=-\infty}^{\infty} \tilde{a}_i \cdot rect\left(\frac{t-iT}{T}\right) \tag{2.9}$$

where the *rect* function is the well-known rectangular pulse function [7].

2.2.6 Pulse Shaping

The Fourier transform of the *rect* function as in 2.9 is an infinite length *sinc* function, meaning that an infinite bandwidth is required for reproduction of a signal of the form of 2.9. Thus *pulse shaping* is introduced; the shape of the pulse used to transmit data determines the characteristics of the signal in both the time and the frequency domains.

In the digital method used in this thesis to achieve transmit pulse shaping, the symbol stream $\tilde{a}_i$ is padded with zeros such that a number of zeros, determined by an *oversample* rate n, are placed between consecutive $\tilde{a}_i$'s. A symbol stream $\tilde{a}_i$ oversampled by a rate of $n = 4$ would yield a series $\tilde{a}'_k$ shown in equation 2.10.

$$\tilde{a'}_k = \{\tilde{a}_1, 0, 0, 0, \tilde{a}_2, 0, 0, 0, \tilde{a}_3, 0, 0, 0, \tilde{a}_4, \ldots\} \tag{2.10}$$

where the sample rate of the series $\tilde{a}'_k$ is a n times the symbol rate.

Each of the $\tilde{a}_i$'s in 2.10 is now considered to be an *impulse*. To apply a pulse shape of $h_{TX}(t)$ to the series of 2.10, the $\tilde{a}'_k$ series is convolved with $h_{TX}(t)$ resulting in a baseband signal defined as

$$\tilde{s}(t) = \sum_{k=-\infty}^{\infty} \tilde{a}'_k \cdot h_{TX}(t - kT) \tag{2.11}$$

where $h_{TX}(t - iT)$ is the desired symbol pulse shape, T is the symbol period and $\tilde{a}'_k$ is the oversampled symbol stream.

A commonly used pulse shape and the one employed in this thesis is that of the *square-root raised cosine* (SRRC) pulse, which may be written as

$$h_{SRRC}(t) = 4\alpha \cdot \frac{\cos\left[(1+\alpha)\pi t \,/\, T\right] + \frac{\sin[(1+\alpha)/T]}{4\alpha t/T}}{\pi\sqrt{T}\left[1 - \left(r\alpha t \,/\, T\right)^2\right]} \tag{2.12}$$

Typically the receiver will use a receive filter *matched* to the transmit pulse, forming a *matched filter* pair. When the two matched filters are convolved the product is a response

known as a *raised cosine* pulse function, as described in [9] and defined in the time domain by

$$h_{RC}(t) = sinc\left(\frac{\pi t}{T}\right) \cdot \left(\frac{\cos\left[\pi\alpha \cdot t/T\right]}{1-4\cdot\left[\alpha \cdot t/T\right]^2}\right) \tag{2.13}$$

where T is the symbol period and the variable α is the *roll-off* factor that determines the transmission bandwidth.

The SRRC roll-off parameter α is a means of explicitly specifying the amount of excess bandwidth over that of the ideal frequency domain solution of a rectangular spectral envelope. The relationship between bandwidth and rolloff parameter α is given by

$$B_T = \frac{1}{T}(1+\alpha) \tag{2.14}$$

where B_T is the resultant bandwidth and T is the symbol period.

An attractive quality of this pulse shape is that when in perfect timing synchronisation the function has zero crossings at periodic (iT) sampling instances other than that of the main lobe. These then make no contribution to inter-symbol interference (ISI) and the pulse satisfies the Nyquist criterion [10].

In a typical digital modem, the pulse-shaping filter is implemented as a Finite Impulse Response (FIR) filter structure, where the length of the filter as defined by 2.13 is truncated in the time domain. Figure 2.4 is a plot generated for a truncated raised cosine pulse generated by 2.13 with varying roll-off factors

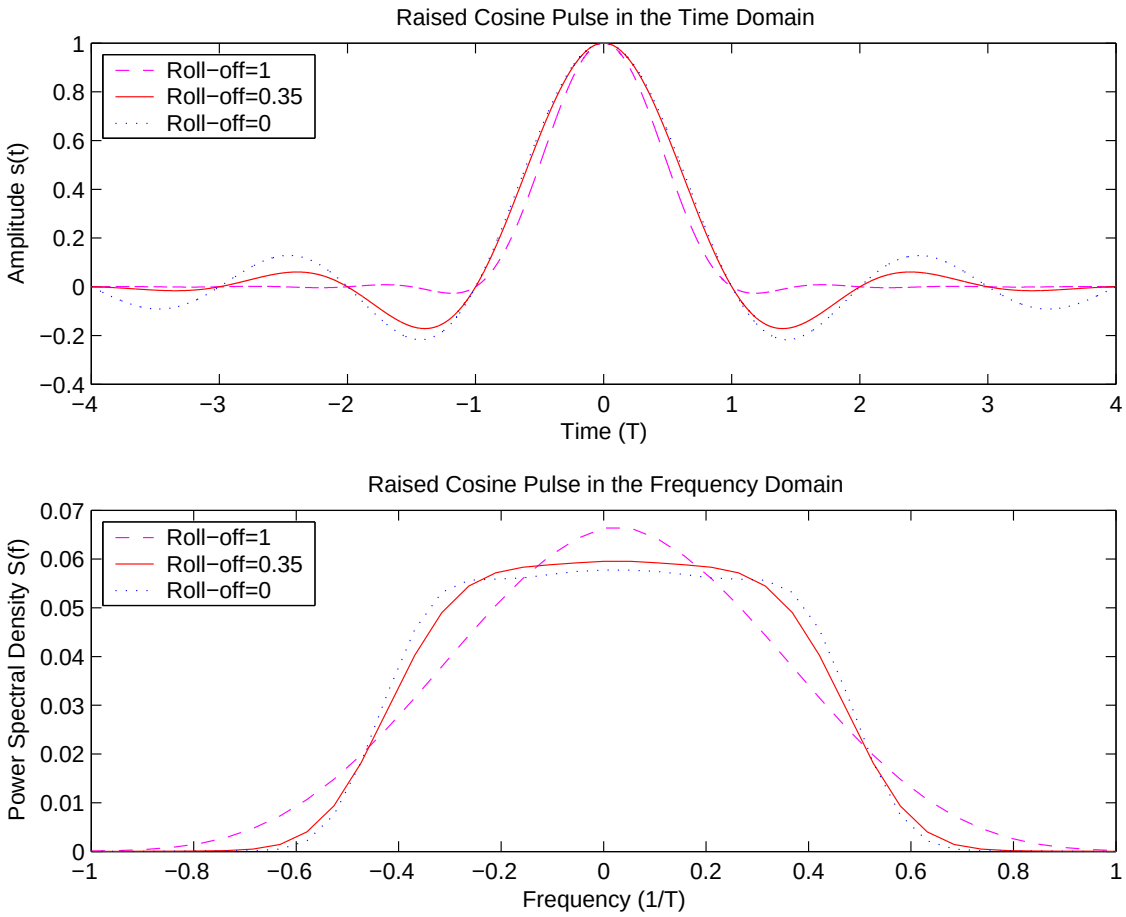

Figure 2.4: Truncated Raised Cosine Pulse, Truncated to 8 Symbol Intervals

where the pulse extends from *-4T* to *4T* and the power spectral density is calculated using the periodogram of a pulse of the same length.

The SRRC pulse shape employed in the simulations of this thesis will have a roll-off of 0.35 and a filter length of *8T* to resemble the raised cosine function shown by the solid line in Figure 2.4. It should be noted that as the pulse is not of infinite length, the frequency response will be spread slightly wider than the spectral envelope of the ideal SRRC pulse.

As the roll off factor is increased towards unity, the fraction of transmitted energy concentrated in the main lobe (the signal energy occurring in the time domain between $-T$ and $+T$) of the pulse shape increases. In practical systems the advantage of less energy located in the side lobes becomes apparent when a receiver is not in time synchronisation, as there is then less energy in the side lobes distorting the adjacent symbol, causing ISI. The rolloff α also has an effect on the performance of the Digital Phase Locked Loop

(DPLL) used for timing recovery, as described in section 3.4.3.

2.2.7 Direct Digital Synthesis

In a practical implementation, including the DSP implementation employed in this thesis, digital hardware eliminates the need for analog translation of a baseband signal to some bandpass signal at a carrier frequency through *direct digital synthesis* (DDS) [11]. DDS is a means of creating a real analog bandpass signal directly from a complex baseband digital signal. It is a digital to analog (D/A) conversion process with the added ability to *synthesize* an analog bandpass signal, as the original digital spectrum is directly translated onto some carrier in the frequency domain.

The transmitter that is implemented in section 5.3.3 makes use of DDS. A purpose built chip [12] properly programmed need only be passed pulse shaped baseband digital data in order to directly generate an analog passband digitally modulated signal.

2.3 Digital Receiver Methods

This section presents methodology for the digital receiver design employed in this thesis. A generic digital receiver structure is presented, along with the process of bandpass sampling, showing how an estimated signal vector is recovered in the coherent case and how the symbol decision is made.

2.3.1 Receiver Structure

The synchronisation subsystem developed in this thesis will form a component of the experimental space-time wireless platform *SASRATS* that is under ongoing development at Canterbury University [5]. The SASRATS receiver employs the generic structure illustrated by figure 2.5. This structure is used in the simulations of this thesis in chapter 4 as well as the practical implementation of chapter 5. Digital receivers such as this avoid the need for analog demodulation to baseband by taking advantage of passband sampling [13] which is briefly described in section 2.3.2.

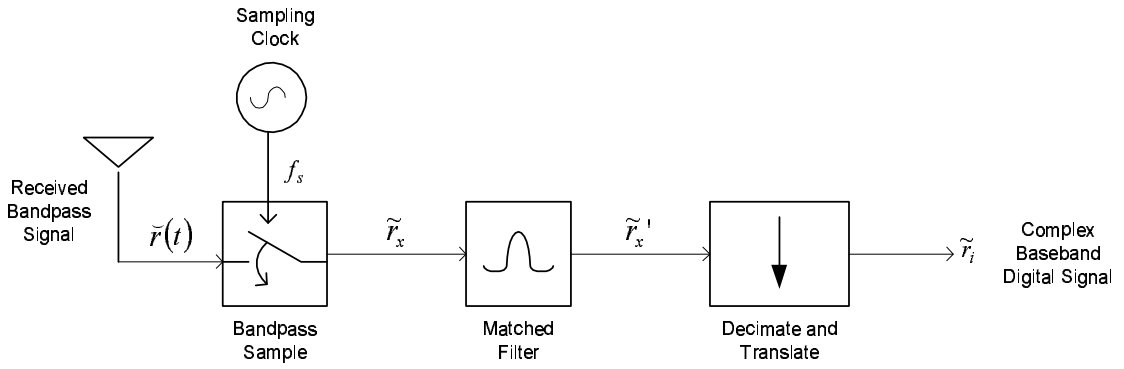

Figure 2.5: The Front End of a Digital Receiver

The receiver front end illustrated in 2.5 assumes that sampling is *synchronised* to the symbol phase. In the case where signal timing is known, the complex sample sequence $\tilde{r}_i$ is expressed by.

$$\tilde{r}_i = \sum_{i=-\infty}^{\infty} \tilde{a}_i \cdot \delta(iT - t) + n(iT) \tag{2.15}$$

2.3.2 Bandpass Sampling

Bandpass sampling is a digital technique, where sampling is specified at some rate that is less than the *Nyquist rate* for a given bandpass signal whilst maintaining the spectral integrity of the bandpass of frequencies of interest. Possible configurations and rules of passband sampling are presented in [13]. This technique takes advantage of spectral *aliases*[1] or *images* of the sampled spectrum that are repeated in the digital frequency domain. These images occur as a function of the pulse train spectrum of the sampling process being convolved with that of the spectrum of interest, as illustrated in Figure 2.6.

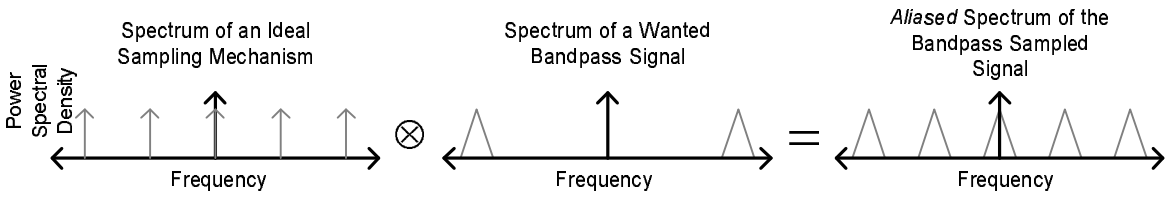

Figure 2.6: Bandpass Sampling: Convolution in the Frequency Domain

[1] A spectral *alias* should not be confused with *aliasing*, which is a form of distortion that arises when poorly configured sampling causes an overlapping of some signal with a wanted spectral image.

where the $\otimes$ symbol denotes a convolution operation.

Following the bandpass sampling process of figure 2.6, the spectral alias that occurs at baseband (zero frequency) of the resultant signal may then be isolated by means of a low pass filter.

For a bandpass signal defined with lower and upper frequencies of f_L and f_U respectively giving a bandwidth $B = f_U - f_L$, the following inequalities below must be satisfied to avoid spectral aliasing,

$$\frac{2f_U}{n} \leq f_s \leq \frac{2f_L}{n-1} \tag{2.16}$$

where n is an integer such that

$$1 \leq n \leq I_g\left[\frac{f_U}{B}\right] \tag{2.17}$$

and $I_g\lceil\cdot\rceil$ denotes a *largest integer less than* function.

2.3.3 Recovery of Signal Vector

Optimum receiver principals are discussed in [6], where the geometry of multi-dimensional signal vectors is exploited to produce symbol estimates. For QPSK signal vector recovery, a signal vector $\left\{r^I, r^Q\right\}$ can be formed using the basis functions of equation 2.2 to extract the co-ordinate values as

$$\begin{aligned} r_i^I &= \int_{iT}^{(i+1)T} \check{r}(t)\cdot\varphi^I(t)dt \\ r_i^Q &= \int_{iT}^{(i+1)T} \check{r}(t)\cdot\varphi^Q(t)dt \\ & iT \leq t \leq (i+1)T \end{aligned} \tag{2.18}$$

where $\check{r}(t)$ is the received bandpass signal.

The receiver then produces complex baseband samples using the process as shown in

figure 2.5

$$\tilde{r}_i = r_i^I + j \cdot r_i^Q \tag{2.19}$$

2.3.4 Decision Regions

A received vector $\tilde{r}_i$ as defined by equation 2.19 can be interpreted as a geometric vector, where the signal components r_i^I and r_i^Q form the elements of a two dimensional vector. This geometrical interpretation of the signal $\tilde{r}_i$ leads to the definition of *decision regions* as a method of estimating the most likely transmitted symbol [6]. The decision region for a received signal vector is based on the closest *Euclidean* distance to one of the possible transmitted symbols. For a two dimensional signal space constructed with the basis functions φ^I and φ^Q a QPSK alphabet of four symbols maps to the decision regions as represented in figure 2.7

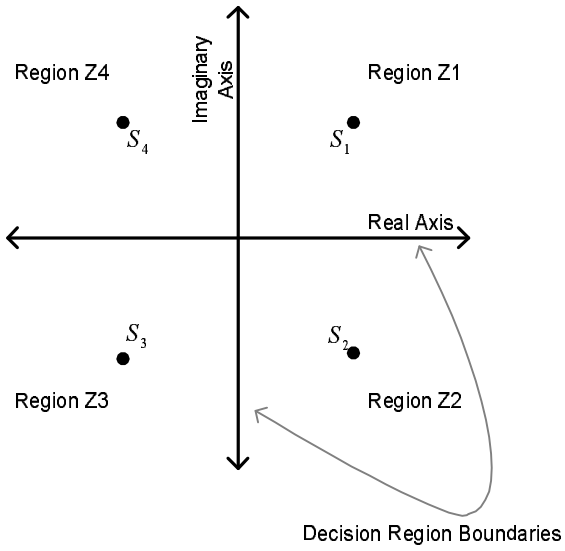

Figure 2.7: Decision Boundaries of Received Signal Vector $\tilde{r}_i$

For a received vector, say $\vec{\rho}$, the decision regions define the possible areas in which vector $\vec{\rho}$ may occur. An occurrence of $\vec{\rho}$ in a given region $\{Z1, Z2, Z3, Z4\}$ then maps $\vec{\rho}$ to

a most likely or closest transmitted vector $\left\{ \ (1,1), \ \ (1,-1), \ \ (-1,-1), \ \ (-1,1) \ \right\}$ or corresponding symbol $\hat{\hat{a}}_i = \{1+j, 1-j, -1-j, -1+j\}$.

2.4 Mobile Communications Channels

This section provides both a channel description and a channel modeling method for the intended operating environment. The term *mobile* is included in this label because either the transmitter or receiver is often in motion, i.e. a mobile phone user conducting a phone call whilst driving or walking. In the case of a typical mobile phone network, a mobile handset communicates from within a cluttered environment to a fixed base station. The effect of this clutter is to provide multiple reflected or diffracted paths between each transmitter and receiver, often creating a combination of a line of sight (LOS) path with one or several non line of sight (NLOS) paths. The presence of the NLOS paths will cause fluctuations in the path loss between the base station and mobile, resulting in signal *fading*. The work of [14, 15] discusses fundamental issues related to radio propagation that are summarised below.

The effect of path loss due to the distance between transmitter and receiver is a trivial case, modeled simply in the free space path loss case by an inverse square law. Loss due to pathway obstructions, known as *long-term fading* or *shadowing*, is a non-trivial case in practice requiring empirical path loss modeling techniques, such as [16]. In these cases the change of path loss varies slowly with time, effectively dictating the mean level of signal available at the receiver and ultimately determining the average signal to noise (SNR) ratio.

2.4.1 Short Term Fading

Short term fading is described in [14]. It is the result of a transmitted signal following two or more paths. The received components then combine destructively and/or constructively depending on the delays involved. These paths are subject to change in time, typically due to mobility of the receiver. This is called *multipath* propagation. Figure 2.8 below illustrates a multipath channel, which may be characterised by the time varying impulse response $h_n(t,\tau)$

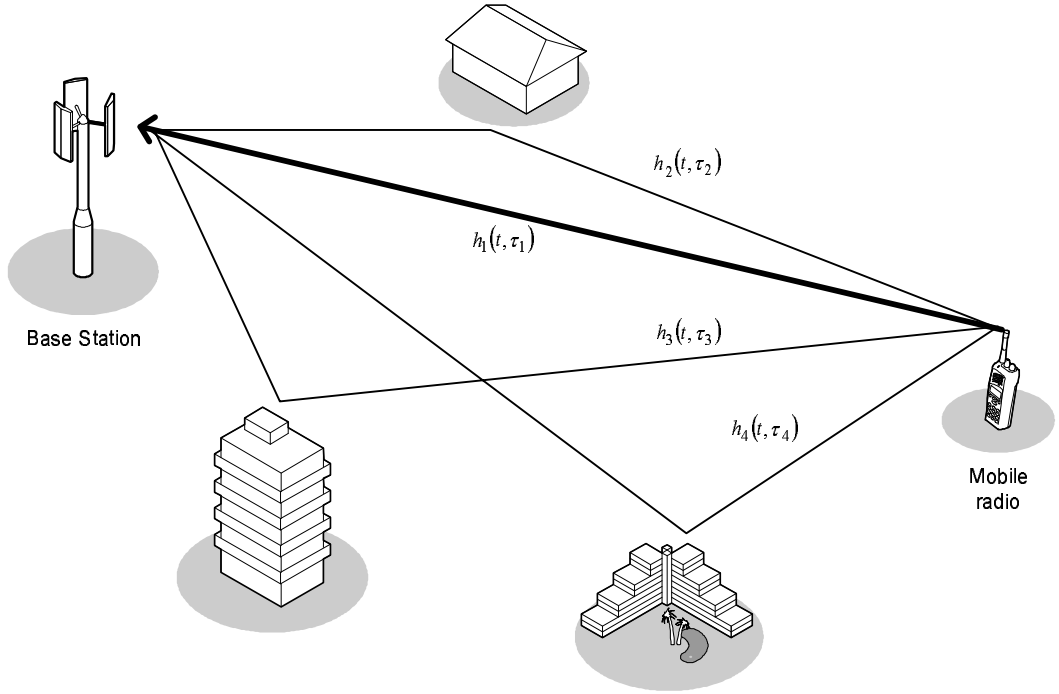

Figure 2.8: Multipath Channel

where the path delays and incident phase angle are functions of the different distances that the signal travels along each path.

For a transmitted signal $\check{s}(t)$, the received signal $\check{r}(t)$ is then the sum of transmitted signal convolved with each path $h_S(t,\tau_n)$ for its given delay.

$$\check{r}(t) = \sum_{s=1}^{S} \int_0^{\infty} \check{s}(\tau_n) \cdot h_s(t,\tau_n)\, d\tau \tag{2.20}$$

where S is the number of paths that comprise the received signal $\check{t}(t)$.

The received signal strength is then the sum of the short-term fading components effectively superimposed onto the long-term fading component, adding a zero-mean probability distribution function to the existing long term fading function.

When short term fading arises only as a sum of NLOS components, then the resultant received envelope will typically obey a Raleigh distribution [17]. When there is a LOS path present as well as NLOS components then the received envelope usually follows a Rician distribution [17].

2.4.2 Multipath Channel Characteristics

Doppler shifts arise as a function of relative motion between the transmitter and mobile station. The Doppler frequency is defined in [13] as

$$f_d = \left(\frac{v}{\lambda}\right)\cos\theta \tag{2.21}$$

where v is the velocity of the mobile station, λ is the wavelength and θ specifies the component of the velocity vector of the mobile that is in parallel with the incident wave.

Table 2.3 gives maximum design Doppler frequencies (with $\theta = 0^\circ$) for the GSM900 band for a maximum utilized downlink frequency of 960MHz and for the maximum uplink frequency of 915MHz.

Velocity (km/h)	Max Doppler Uplink (Hz)	Max Doppler Downlink (Hz)
8	6.6	7.1
50	41.2	44.5
100	82.5	89.0

Table 2.3: Maximum Doppler Frequencies

The *Doppler power spectrum* described in [9] models[2] the frequency domain characteristics of a Raleigh distributed multipath mobile channel as a function of Doppler frequency. The Doppler power spectrum is defined as

$$d(f) = \begin{cases} \frac{1}{2\pi f_d} \cdot \frac{1}{\sqrt{1-\left(\frac{f}{f_d}\right)^2}} & |f| \leq f_d \\ 0 & |f| > f_d \end{cases} \tag{2.22}$$

For the maximum Doppler frequency in table 2.3 of 89Hz, the *Doppler power spectrum* as calculated by 2.22 is shown in Figure 2.9.

[2] Assumes isotropic scattering at the receiver

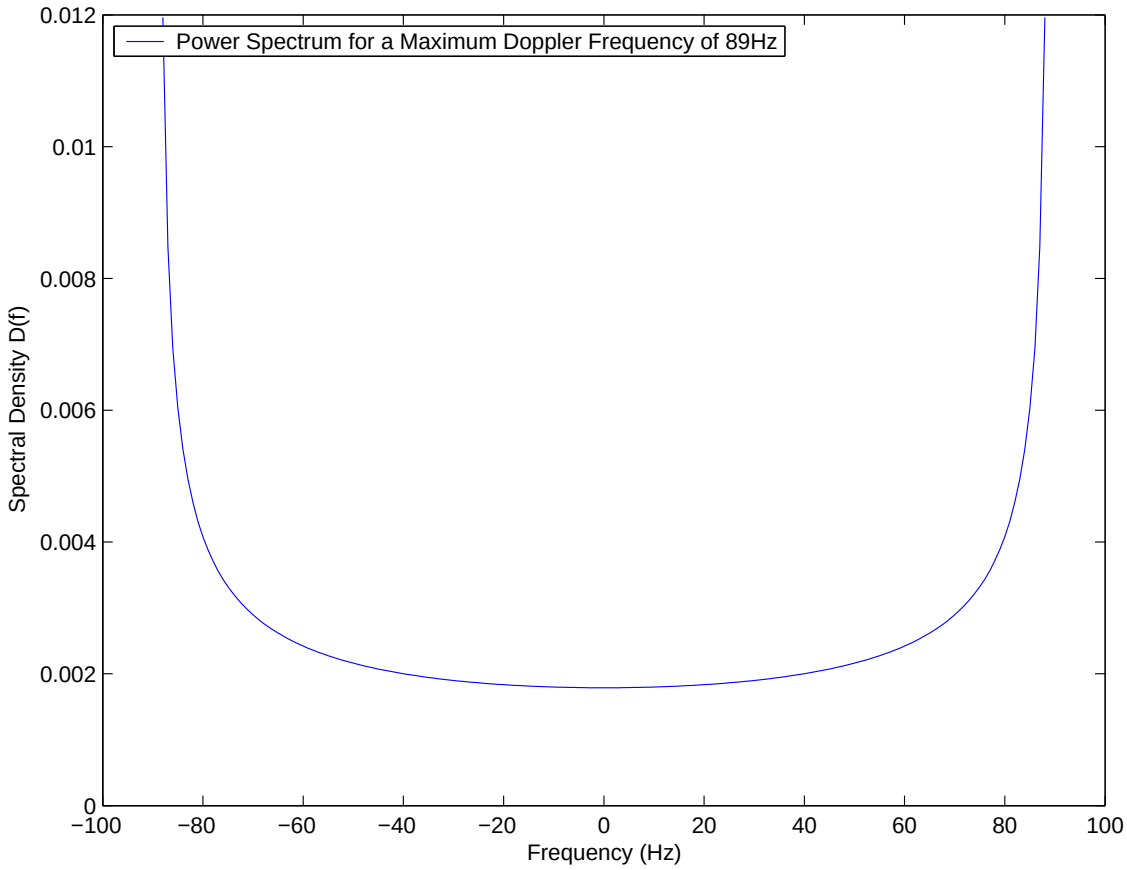

Figure 2.9: Doppler Spectrum for a Mobile Radio Channel

Time delay spread as the result of multipath propagation is defined in [18]. For a transmitted impulse, the actual pulse shape incident at the receiver arising from the varying arrival times of signals traveling via different length paths has a spread in time known as the *delay spread*. When the symbol period is comparable or less to the delay spread, the resulting overlapping of symbols gives rise to inter-symbol interference (ISI) also known as *dispersion*, that results in an *irreducible* error floor[3]. ISI effectively behaves as an additional signal dependent noise source.

Coherence Bandwidth is the upper limit on the data bandwidth that can be used before the fading process becomes *frequency selective*. The coherence bandwidth in a urban street level environment is calculated in [19] to be in the range[4] 14-100MHz. The co-

[3] Termed *irreducible,* ISI cannot be reduced by increasing signal power, but can be compensated for by channel equalisation

[4] [19] Calculates B_c based on multi-ray optical theory and Geometrical Theory of Diffraction (GTD) for outdoors near street level at 28.8GHz.

herence bandwidth can be expressed [20] with an inverse relationship to the delay spread as

$$B_c \approx \frac{1}{2\pi\Delta t} \tag{2.23}$$

where Δt is the delay spread.

Coherence time is a measure of the rate of change of channel characteristics. Coherence time is expressed empirically in [21] as an inverse function of Doppler frequency as

$$T_c \approx \frac{2\pi}{5 f_d} \tag{2.24}$$

When fading occurs faster than the symbol rate, i.e. when the symbol period is longer than the coherence time then the fading process is termed *fast fading*. In the case where the symbol period is significantly shorter than the coherence time the fading process is *slow fading*.

2.5 Simulation of a Communication Channel

In this thesis a channel model will be employed that includes the effects of both the path loss of a communications channel and the short term fading process. For the purposes of simulation, signals are represented as discrete complex series at baseband frequencies. The sampled, complex received signal $\tilde{r}_i$ is defined at baseband by

$$\tilde{r}_m = \tilde{c}_m \cdot \tilde{s}_m + \tilde{n}_m \tag{2.25}$$

where $\tilde{s}_m$ is the transmitted signal, $\tilde{n}_m$ is complex additive white Gaussian noise (AWGN) and $\tilde{c}_m$ is multiplicative distortion generated by the fading process.

2.5.1 Gaussian Noise Sequence

The complex additive Gaussian noise sequence $\tilde{n}_m$ is generated as shown in figure 2.10

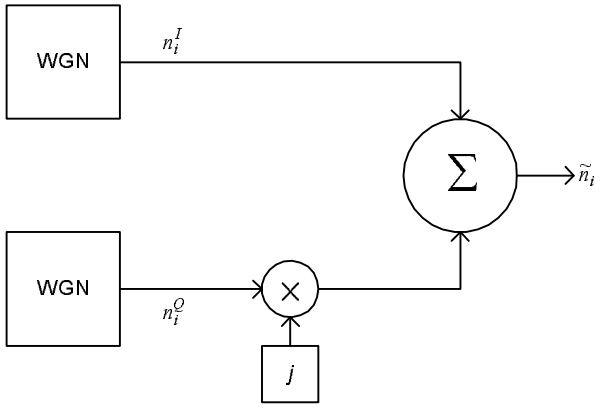

Figure 2.10: Complex Gaussian Noise Generation

where n_m^I and n_m^Q are Gaussian distributed and zero mean with the noise power of the signal determined by

$$\sigma^2 = E\left[|\tilde{n}|^2\right] = E\left[n_I^2\right] = E\left[n_Q^2\right] \tag{2.26}$$

The simulated signal to noise ratio (SNR) is set to a predetermined level by first generating a complex noise sequence $\tilde{n}_m$, and then scaling as

$$\tilde{n}_m = \tilde{n}_m \times \frac{E\left[|\tilde{s}_m|^2\right]}{\sigma^2 \cdot 10 \cdot exp\left[\frac{SNR}{10}\right]} \tag{2.27}$$

thus the signal to noise ratio can be explicitly defined by the variable *SNR*.

2.5.2 Fading channel Sequence

A method[5] for modeling the complex channel series $\tilde{c}_m$ is described in [22] and depicted in figure 2.11. The fading channel is simulated by passing Gaussian noise through a pair

[5]This method does not assume isotropic scattering at the receiver as described in the Doppler power spectrum model of section 2.4.2, but is a useful model that approximates many real scenarios [10].

of 3rd order low pass Butterworth filters, each with a -3dB point at the required Doppler frequency f_d. The filtering partly acts to correlate the random processes and the cut-off frequency dictates the level of correlation and the rate of fading.

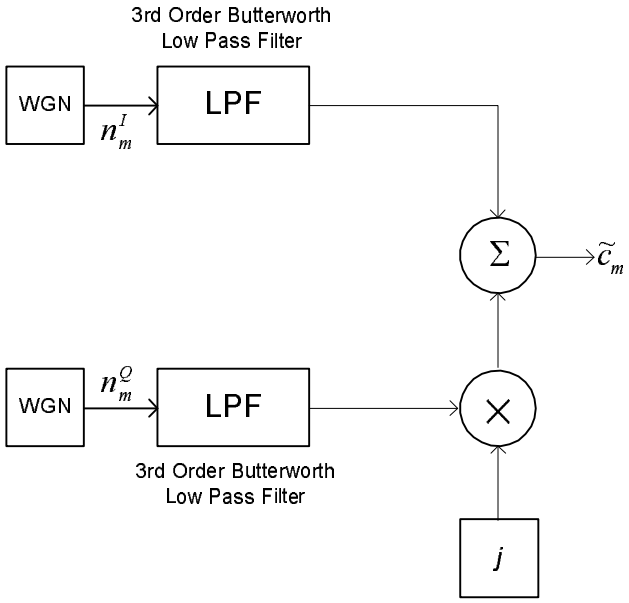

Figure 2.11: Fading Channel Sequence Model

where the average power of the fading envelope is normalised such that

$$E\left[|\tilde{c}_m|^2\right] = 1 \tag{2.28}$$

The requirement of 2.28 means that the average power transfer of the channel is unaffected by the fading, so that the required average SNR can then be set solely as a function of the noise power.

Figure 2.12 depicts the in-phase and quadrature components of a slow fading channel generated using the model illustrated in figure 2.11.

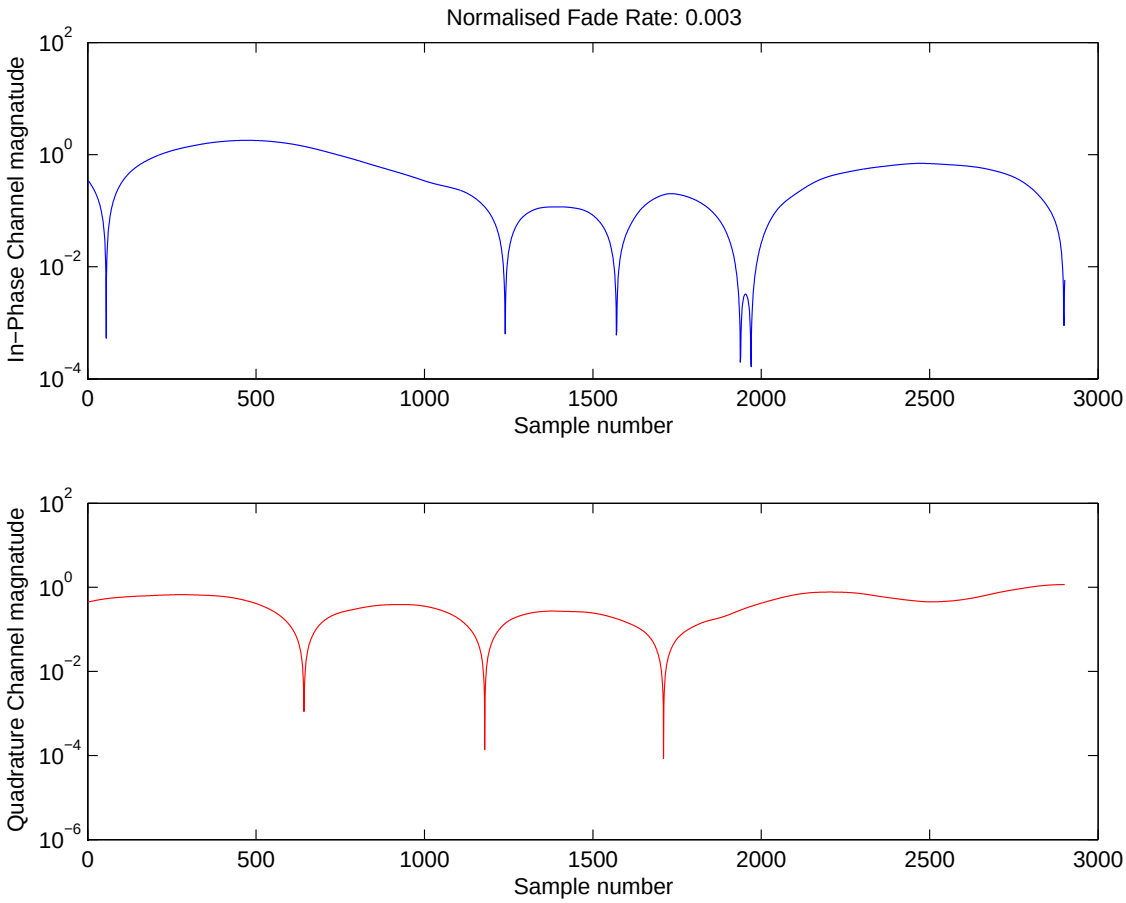

Figure 2.12: In-Phase and Quadrature Magnitudes of a Slow-Fading Channel

where the *normalised fading rate* f_{dT} is the Doppler fading frequency divided by the sampling frequency

$$f_{dT} = f_d \cdot T \tag{2.29}$$

and T is the symbol period.

2.6 Summary

In this chapter some fundamentals of digital communications systems relevant to the remaining chapters of this thesis have been summarised. The fundamentals of this chapter are intended to provide background for the development of a digital phase locked loop

(DPLL). They are also intended to provide a theoretical basis for the test environment in which the real-time prototype DPLL was tested.

The representation of vector signals and the implementation of D-QPSK coding and modulation is pertinent to the type of signal the DPLL will receive and process. AWGN and flat fading channels are described and the modeling processes discussed. These will be employed for receiver testing. The actual structure of the digital receiver and digital receiver methods are critical to the implementation of the DPLL in an actual prototype.

Chapter 3

Receiver Timing Synchronisation

This chapter presents necessary background on the design and operation of analog phase locked loops (PLLs) and digital phase locked loops (DPLLs). First the concept of *timing phase error* is introduced. In a typical, rather than simulated system, clock timing cannot be *perfectly* reproduced by independent transmitter and receiver systems. A PLL or DPLL is a means of mitigating the phase error by producing an estimation of phase error and applying compensation to *acquire* then and *lock* onto the phase of the received signal.

This chapter continues to describe how digital technology presents the opportunity for new types of receiver structures to be designed, reducing the analog processing requirement in a receiver with functionality transposed into the more flexible digital domain. Such a structure is employed in the DPLL implemented in this thesis.

3.1 Timing for Signal Sampling and Symbol Estimation

For the purpose of explaining digital synchronisation, it is convenient to assume access to samples of the received signal in complex baseband.

The analog form of an (in-phase or quadrature) component of a baseband QPSK waveform is illustrated in figure 3.1. It is a signal of this form that will be sampled and then processed by the DPLL in order to produce an estimate of the original symbol. The vertical lines have the same periodicity as the original symbol timing and are presented to represent the timing for optimal symbol recovery.

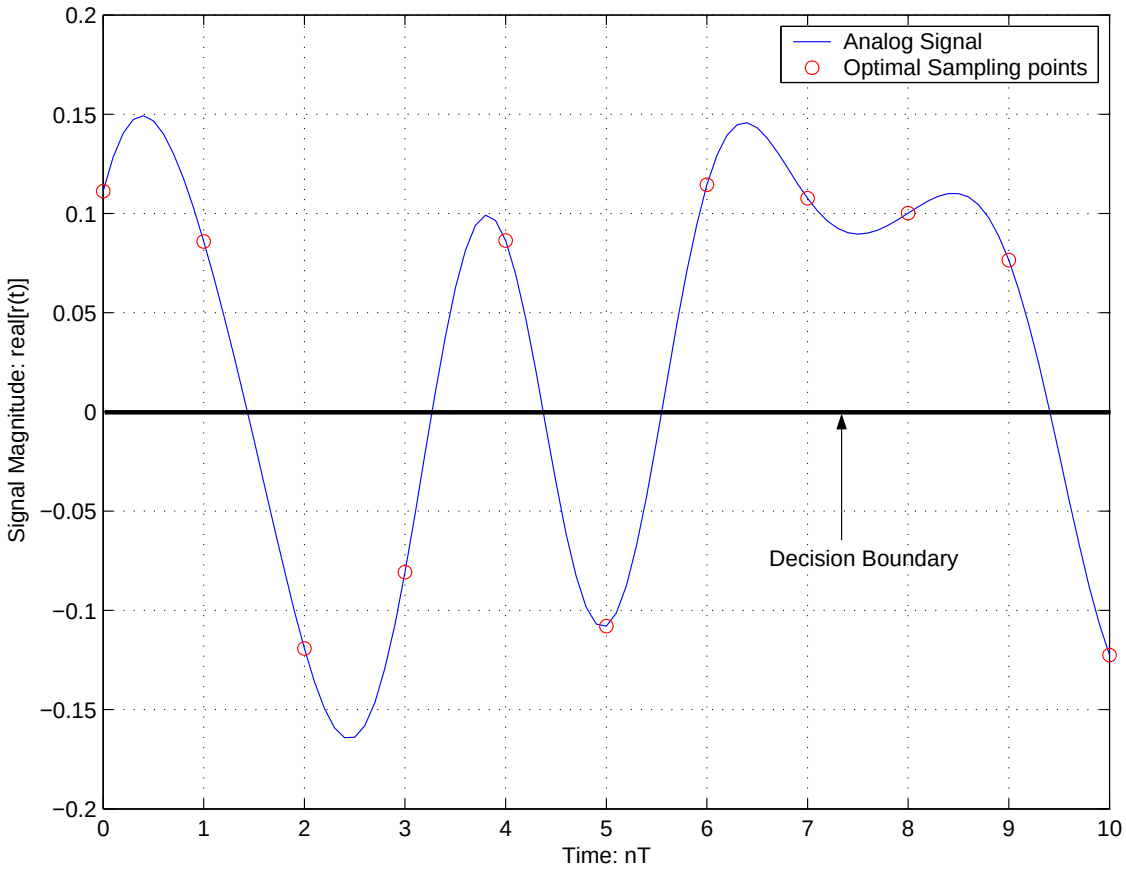

Figure 3.1: A Baseband analog response for a component of a QPSK signal showing optimal sampling timing; pulsed shaped with a SRRC pulse, rolloff of 0.35, corrupted by AWGN of 10dB SNR channel and match-filtered. T is the symbol period.

Figure 3.1 is included to assist with a visualisation of the analog signal source from which digital data is to be extracted, and is typical of a non-faded signal employed; band limited by an SRRC filter and corrupted by AWGN. The optimal timing (or phase) at which to estimate the transmitted symbol is not readily evident.

When the signal of figure 3.1 is plotted with the time base modified to a multiple of *t mod T,* it forms a plot commonly known as an *eye diagram.* An eye diagram is useful for observing the effects arising from noise and interference on the received signal, and in this case to illustrate the reduced ability to discriminate between symbols due to imperfect sample timing. An eye diagram for the signal of figure 3.1 is illustrated in figure 3.2.

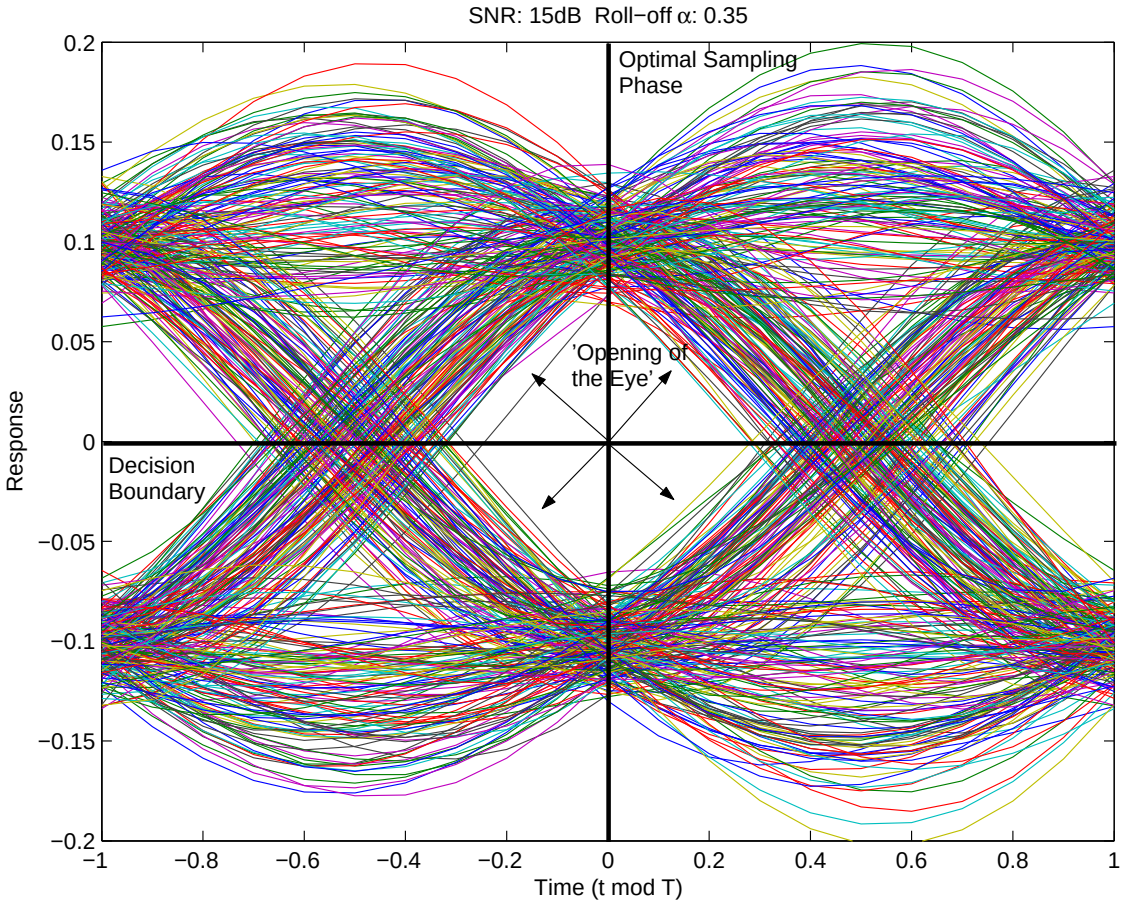

Figure 3.2: Eye Diagram, for a baseband SRRC pulse with roll-off of $\alpha = 0.35$ over a AWGN channel of SNR=15dB. This plot produced by MatLab using channel simulation methodology described in section 2.5.

What is evident from figure 3.2 is there is the widest gap between the positive pulses, negative pulses and the decision boundary at the timing point $t = 0$, which is the point of *maximum discrimination* between possible symbols. It is the knowledge of this timing or phase that is critical for a receiver to recover in order to achieve optimal performance. The space between where the signals cross the decision boundary is sometimes referred to as *the opening of the eye*.

A variable ε is now defined as the difference in time between the actual sampling phase and the optimal sampling phase. Equation 2.15 from chapter 2 then becomes

$$\tilde{r}_i = \sum_{i=-\infty}^{\infty} \tilde{a}_i \cdot \delta(iT - t + \varepsilon) + n(iT + \varepsilon) \tag{3.1}$$

The timing error is another source of corruption of the reception of received data. The relationship for a received signal $\tilde{r}_i$ modeled by equation 3.1 between timing error ε and the resultant BER is illustrated by figure 3.3.

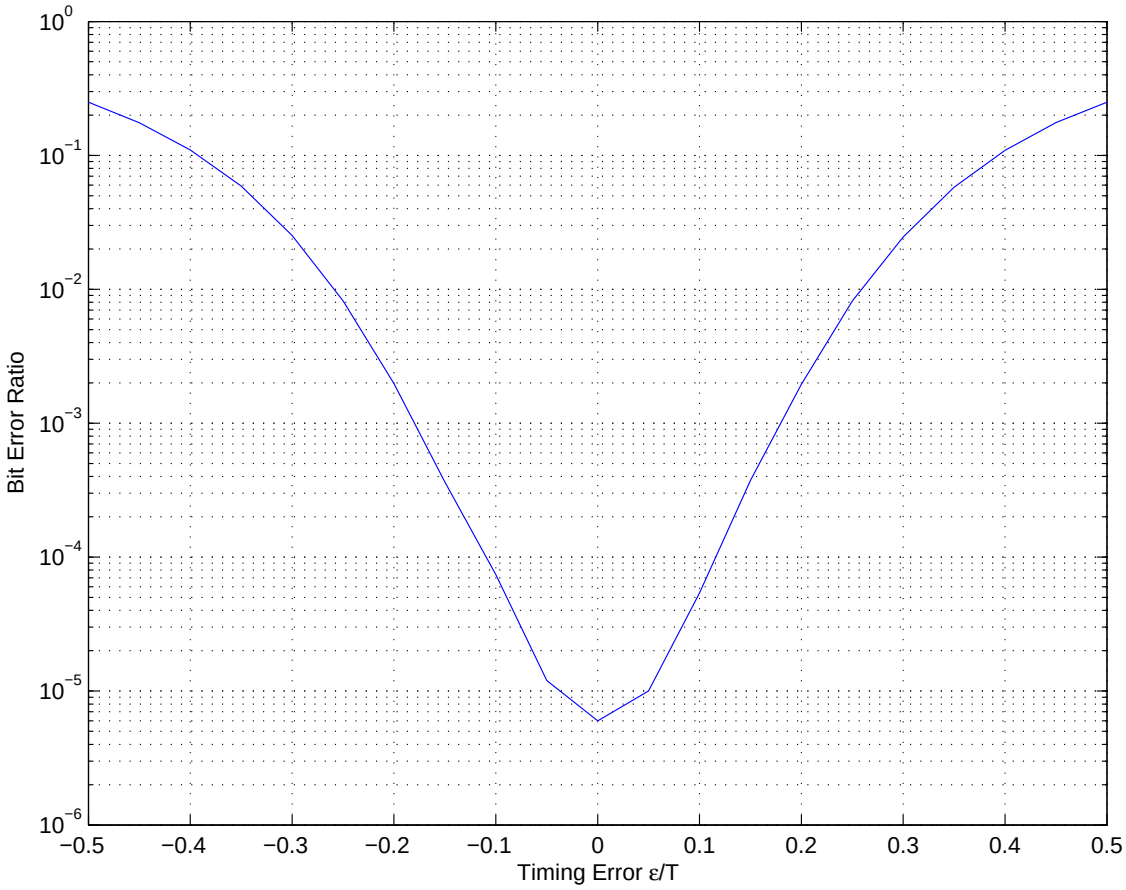

Figure 3.3: BER as a function of timing error ε. Channel is corrupted by AWGN with a SNR of 7dB and a raised consine pulse shape with a rolloff of 0.35

It is evident from figure 3.3 the effect that timing error ε has on the BER performance. The timing window for near-optimal performance is approximately $\pm 0.1T$, after which any increase in $|\varepsilon|$ has the effect of rapidly increasing the BER. This increase in $|\varepsilon|$ has the ultimate effect of increasing the BER to a value 0.5 in the case of BPSK or QPSK.

3.2 Timing Recovery in the Analog Domain

As discussed in [7] a *phase-locked loop* (PLL) is a *negative feedback* system. An analog signal uses a voltage level to proportionally represent the phase error of the system and

to adjust a Voltage Controlled Oscillator (VCO) to have the same phase angle[1] as the received signal. A basic PLL is shown in figure 3.4.

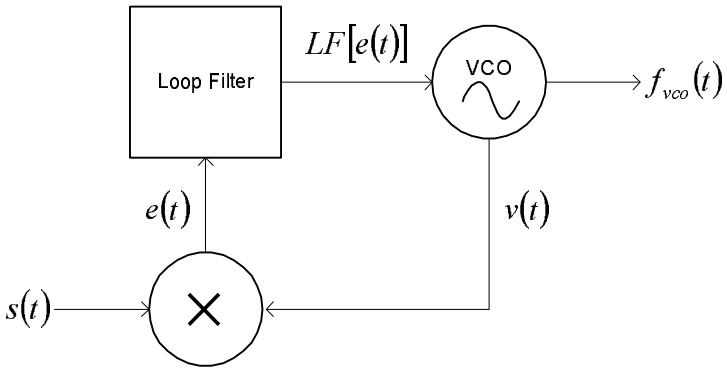

Figure 3.4: PLL Model

Suppose an input signal $s(t)$ is defined by

$$s(t) = A_c \sin[2\pi f_c t + \phi_1] \tag{3.2}$$

and the signal $v(t)$ at the output of the VCO is defined by

$$v(t) = A_v \cos[2\pi f_c t + \phi_2] \tag{3.3}$$

The multiplier[2] $s(t)$ and $v(t)$ produces the signal $e(t)$ as[3]

$$\begin{aligned} e(t) &= s(t) \cdot v(t) \\ &= A_c A_v \frac{1}{2}[\sin[(2\pi f_c t + \phi_1) - (2\pi f_c t + \phi_2)] + \\ &\quad A_c A_v \frac{1}{2}[\sin[(2\pi f_c t + \phi_1) + (2\pi f_c t + \phi_2)] \end{aligned} \tag{3.4}$$

which reduces to a high frequency term

$$e^H(t) = A_c A_v \frac{1}{2}[\sin(4\pi f_c t + \phi_1 + \phi_2)] \tag{3.5}$$

[1] In some receivers (such as an FM receiver) the VCO is designed to produce a signal with an angle offset of 90°.

[2] A *multiplier* of two sinusoids in this context is also known as a *mixer* or *modulator*

[3] Applying trigonometric identity $\sin\alpha\cos\beta = \frac{1}{2}[\sin(\alpha-\beta)+\sin(\alpha+\beta)]$

and a low frequency term

$$e^L(t) = A_c A_v \frac{1}{2}[\sin(\phi_1 - \phi_2)] \tag{3.6}$$

The loop filter filters out the high frequency component, resulting in a VCO steering function $e^L(t)$. This function provides the VCO with a negative feedback signal that steers the VCO to *acquire* and then *lock* to the phase of the received signal. These are the fundamental building blocks of a *phase-locked* system.

This construction also illustrates how any multiplier (or mixing device) can be used to implement a PLL. Typically a multiplier has a gain coefficient, and equation 3.6 would be modified to

$$e^L = k_m A_c A_v \frac{1}{2}[\sin(\phi_1 - \phi_2)] \tag{3.7}$$

where k_m is the *multiplier gain*.

3.2.1 A Digital Receiver Using Analog Timing Recovery

An *analog recovery* method [23] for a digital receiver is illustrated by figure 3.5. A feed-forward method is used to synchronise the sampling clock $f_{VCO}(t)$ with the incoming analog signal $\check{r}(t)$. This method uses an analog processing method to detect the phase of the received signal and to control the phase of the sampling clock.

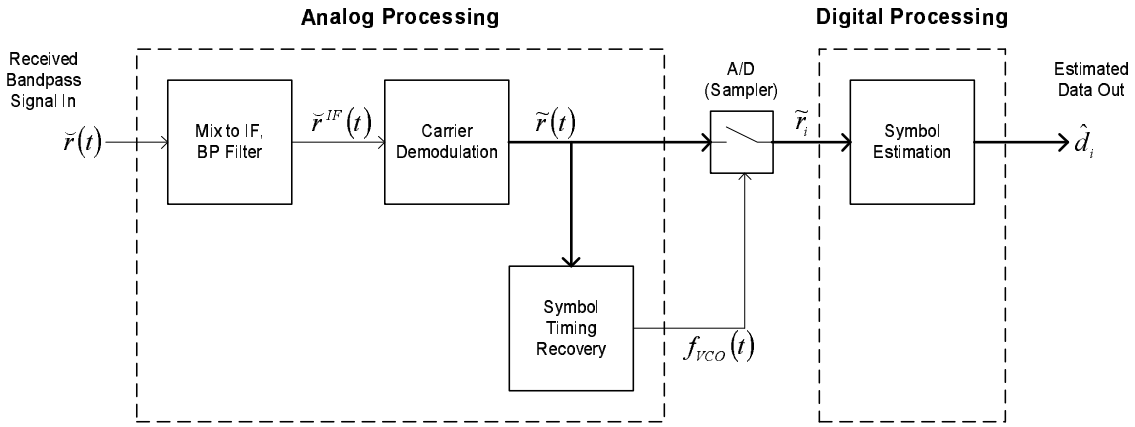

Figure 3.5: Digital receiver front end utilising analog timing recovery: using analog processing to demodulate the received signal and to recover symbol timing

where the received signal $\check{r}(t)$ is a received signal at some passband carrier frequency, $\check{r}_{IF}(t)$ is the signal $\check{r}(t)$ following frequency translation to some fixed *intermediate frequency* (IF) and $\tilde{r}(t)$ is the analog complex baseband representation[4] of $\check{r}_{IF}(t)$.

The first block of the analog processor depicted in figure 3.5 will convert[5] this signal to IF and then filter that signal so as to remove any signal that is not of interest and prevent the possibility of spectral *aliasing* in the sampling process. The IF is a preset frequency that allows the remaining processing to be designed to receive signals at some fixed frequency, typically a lower passband simplifying the analog design and implementation required.

In order to demodulate the signal $\check{r}_{IF}(t)$, the receiver must recover the *carrier phase* of the signal $\check{r}_{IF}(t)$. With the carrier demodulator tracking the carrier phase, the resultant baseband I and Q signals can then be multiplied with the appropriate basis functions[6] to recover the baseband in-phase and quadrature components. The symbol timing of these signals is not known at this point. An analog scheme that performs the demodulation to recover the in-phase and quadrature components is described in section 3.2.2.

In order to estimate the optimal timing at which to sample each of the in-phase and quadrature baseband components symbol timing recovery must be estimated. An early-

[4] As described in section 2.2.5

[5] The *mixer* will be some tuning device that allows the receiver the ability to select the carrier frequency at which to attempt to receive a signal

[6] Such as those described in section 2.2.2

late gate synchroniser [7] as depicted in figure 3.7 illustrates one possible analog scheme that performs this function. With the in-phase and quadrature baseband components recovered, and the optimal symbol timing estimated the analog to digital converter (A/D) now samples the complex baseband signal $\tilde{r}(t)$ to produce a symbol estimate $\tilde{a}_i$, which can then be decoded to estimate the original transmitted data, $\hat{d}_r$.

3.2.2 *Costas Loop* Demodulation of a QPSK signal

A *Costas loop* or *Costas receiver* [24] is a well known means of coherently demodulating a carrier signal in synchronous communications. The Costas loop illustrated in figure 3.6 demodulates a QPSK into its complex components $r_I(t)$ and $r_Q(t)$.

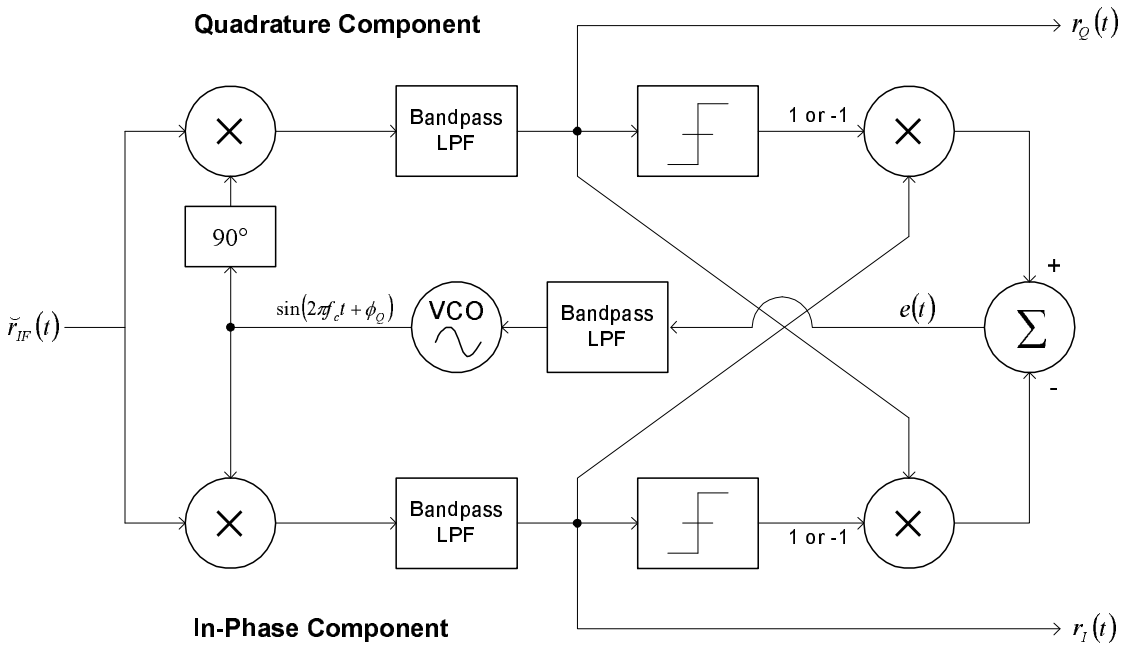

Figure 3.6: Costas Loop for QPSK Demodulation, reproduced from [25]

The mathematics that model the operation are described in [25], and is not considered here. This schematic illustrates an example of the type of analog technique required to process a digitally modulated signal in order to be able to estimate the original transmitted data, d_i.

3.2.3 Early-Late Gate Symbol Synchroniser

The early-late gate symbol synchroniser [7] is one means of estimating the optimal timing to sample each of the baseband signals $r_I(t)$ and $r_Q(t)$ as illustrated in figure 3.7.

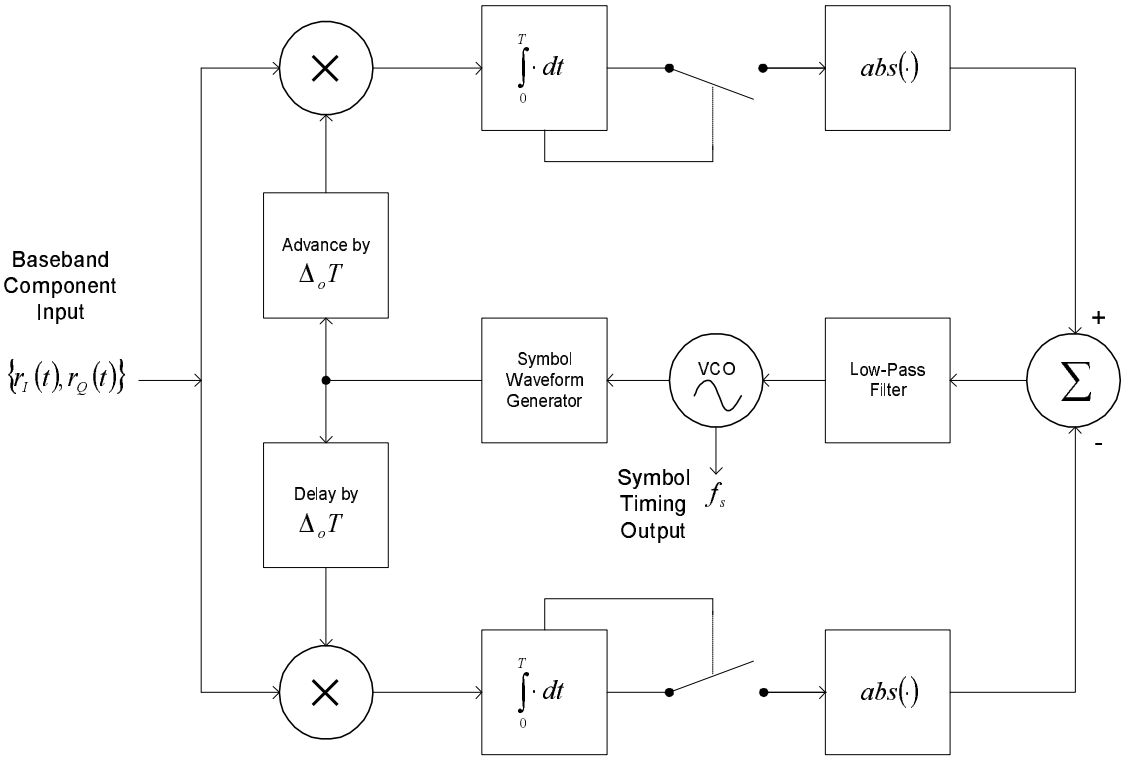

Figure 3.7: An Early-late gate symbol synchroniser, reproduced from [7]

It uses *integrate and dump* blocks to correlate received pulses with those generated at a frequency of f_s generated by a VCO. A feedback loop provides a steering function for the loop that acquires and tracks the phase of received symbols, either[7] $r_I(t)$ or $r_Q(t)$.

As with the Costas loop in section 3.2.2, detailed analysis and performance description of this subsystem is not considered further here. This technique will be referred to in later discussion on the value of digital domain techniques providing an equivalent function.

[7]The timing of $r_I(t)$ and $r_Q(t)$ is the same, so either signal can be used for symbol timing recovery.

3.3 Timing Recovery in the Digital Domain

This section describes the general structure for timing recovery in the *digital domain*. Digital domain techniques of band-pass sampling[8] and frequency translation produce a non-synchronised complex baseband signal. Other digital techniques estimate the timing error and interpolate between the unsynchronised digital signal samples $\tilde{r}_i$, to produce optimal symbol estimates $\tilde{a}_i$.

Consider a comparison of the digital techniques in this section implemented by software against the complexity of the analog processing required to achieve the equivalent reduction of received signal to baseband and estimation of optimal symbol timing. The implementation of analog timing recovery for both carrier and symbol phase acquisition is hardwired and requires component-intensive PCBs, with a fixed design. A software implementation in DSP is capable on the other hand of parameter change through programmable offline adjustment or possibly in real-time. A DSP based board that executes sampling, filtering and other advanced methods can be bought off the shelf, needing only tailoring of relevant software parameters to the requirements.

3.3.1 DPLL Model

A general structure of a *digital recovery* method, is illustrated in figure 3.8,

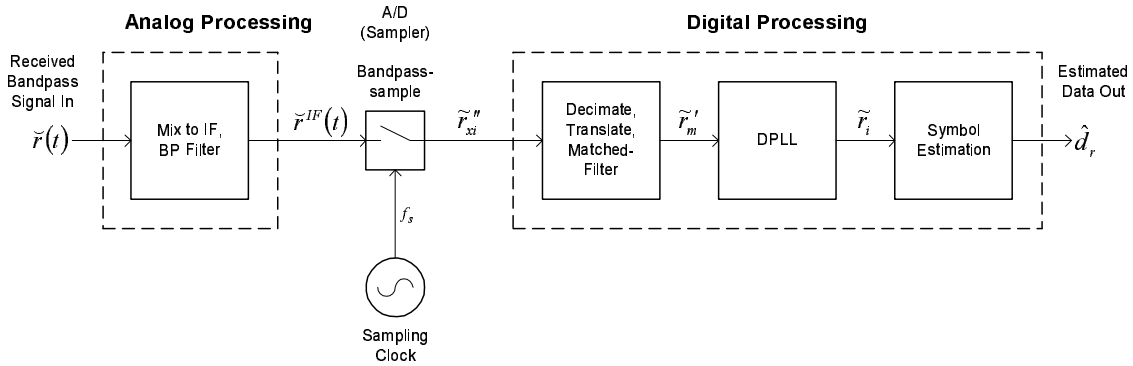

Figure 3.8: Digital receiver front end using digital processing to recover symbols estimates from unsynchronised signal samples

where the output from the A/D block is a digital signal defined as $\tilde{r}''_{xi}$. The 'x' in the

[8]Described in detail in section 2.3.2.

index indicates some arbitrary sample rate, governed by the rules of *bandpass sampling* as described in section 2.3.2. The signal produced at the output of the *decimate, translate and matched-filter* block, $\tilde{r}'_m$ has an index of m indicating a digital signal with a sample rate of at least twice the symbol rate.

The signal processing requirements of the digital structure of figure 3.8 are significantly reduced from the analog approach of figure 3.5. The digital receiver implementation can take advantage of digital techniques such as band pass sampling and eliminate the requirement for the analog signal to be translated to baseband, instead being presented to the A/D block of the receiver at some intermediate frequency (IF).

Digital signal processing can be executed using complex mathematics to estimate timing error, track the phase error and interpolate a symbol estimate at the calculated optimal timing point. It is this interpolation that allows a symbol estimate to be produced at the calculated optimal timing point without the analog signal sampler ever actually sampling the signal at this point. This allows us the ability to utilise a free running clock source that does not have to synchronise with transmitted symbol timing.

For the purposes of this thesis, the receiver model of figure 3.8 is employed for the implementation of the DPLL and the following subsections describe the various design considerations and issues. The DPLL function of the *digital processor* block of figure 3.8 is represented in figure 3.9.

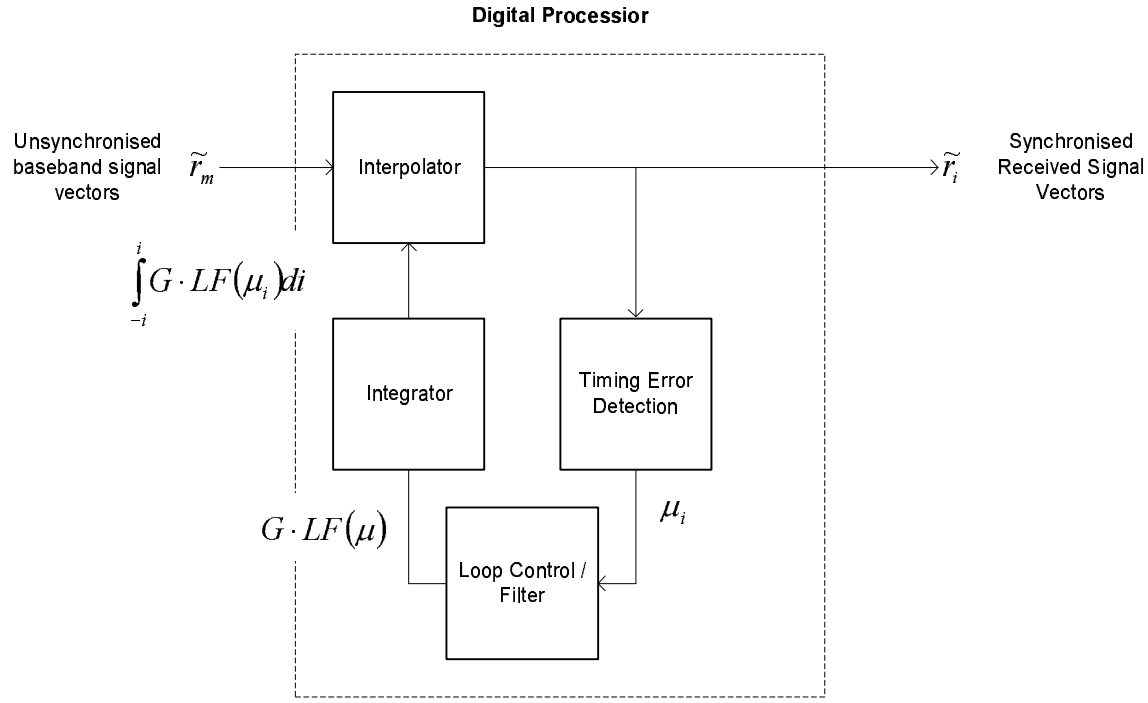

Figure 3.9: Basic overview of the digital processor block

Figure 3.9 illustrates the DPLL sub-module of the digital processor block of figure 3.8. It is these blocks that are of interest; the interpolation block, the Timing Error Detection (TED) block and the loop control / filter block. The design of these blocks is discussed in the following subsections.

3.3.2 Interpolation for Timing Adjustment

The *interpolator* shown in figure 3.9 is effectively a *multi-rate* filter. In this thesis[9] the sample rate f_s is at twice the symbol rate and the filter must *decimate* the incoming samples rate by a factor of 2 in order to match the symbol rate. When the filter performs this decimation, it produces an output that is an estimate of the signal at some estimated timing error, it achieves this by *interpolating* between signal samples at some timing phase offset determined by the Timing Error Detector (TED).

A well known interpolator [26] for the *perfect* recovery of a band-limited signal from

[9]The majority of this research is conducted with f_s set at twice the symbol rate, the preceding research conducted at Canterbury University [8,22] primarily utilised a sampling rate of four times that of the symbol rate.

its samples is achieved by a filter with the impulse response,

$$h_I(t) = \frac{\sin\left(\frac{\tilde{r}_m}{T_s}\right)}{\frac{\pi t}{T_S}} \tag{3.8}$$

This filter would of course require an infinite response in the time domain, so is not realisable. But *perfect* recovery of the original signal is not required in any case [23]. It is only required that the estimated symbol values be correct for data to be *perfectly* recovered.

3.3.2.1 Linear interpolation

In [27] a comparison of the implementation and performance of cubic, linear and piecewise-parabolic interpolation in digital modems is presented. A conclusion of particular interest was that at a low sample rate, 2.5 samples/symbol[10], a linear interpolator could produce a useful level of performance.

The use of a linear interpolator at a low rate was anticipated to allow a significant reduction of the DSP processing required. A linear interpolator is implemented efficiently in DSP with the formula

$$\begin{aligned} \hat{\tilde{a}}_{i+1} &= \tilde{r}_{m+1} + \mu(\tilde{r}_{m+2} - \tilde{r}_{m+1}) \quad 0 < \mu \leq 1 \\ \hat{\tilde{a}}_{i+1} &= \tilde{r}_{m+1} + \mu(\tilde{r}_{m+1} - \tilde{r}_{m}) \quad 0 \geq \mu > -1 \end{aligned} \tag{3.9}$$

where the signals are synchronised such that $\hat{\tilde{a}}_{i+1}$ occurs simultaneously with $\tilde{r}_{m+1}$.

3.3.2.2 Cubic Interpolation

The work of [27] also presented results for the use of cubic interpolation using a *Farrow* structure [28]. The Farrow structure was employed in previous research [8, 22]. The Farrow interpolator can be represented [22] in terms of the matrix multiplication,

[10] *2.5 samples/symbol* relates to the use of either the preceding sample or following sample for the interpolation. The actual sampling rate employed is 2 samples/symbol.

$$\hat{\tilde{a}}_i \;=\; \vec{\mu}\cdot\vec{\vec{A}}\cdot\tilde{\vec{r}}_c \tag{3.10}$$

where the matrices are defined as

$$\vec{\mu}=\begin{bmatrix}\mu^3 & \mu^2 & \mu & 1\end{bmatrix}\quad \vec{\vec{A}}=\begin{bmatrix}\frac{1}{6} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{6}\\ 0 & \frac{1}{2} & -1 & \frac{1}{2}\\ -\frac{1}{6} & 1 & -\frac{1}{2} & -\frac{1}{3}\\ 0 & 0 & 1 & 0\end{bmatrix}\quad \tilde{r}_c=\begin{bmatrix}\tilde{r}_0\\ \tilde{r}_1\\ \tilde{r}_2\\ \tilde{r}_3\end{bmatrix} \tag{3.11}$$

The coefficients of matrix $\vec{\vec{A}}$ in equation 3.11 are those commonly used for *cubic interpolation*. It should be noted that these coefficients can be specifically optimised for use with interpolation implemented for use in a DPLL structure. The work of [29] develops coefficients optimised for DPLL interpolation and the specific pulse shapes employed.

With modern DSP chipsets (such as the Motorola 56321 DSP employed in this thesis) optimised for FIR implementation, matrix mathematics equations like that of 3.10 lend themselves to a reasonably efficient implementation. Both interpolators can be operated at a sample rate of 2 samples/symbol, but the cubic interpolator requires 4 samples in order to calculate each symbol estimate. This thesis explores the use of both cubic (Farrow) and linear interpolation techniques.

3.3.2.3 Comparison of Linear and Cubic Interpolation Techniques

Linear and cubic interpolators offer different levels of interpolated accuracy with a trade-off of processing power. Figure 3.10 illustrates cubic and linearly interpolated responses for a sampled signal typical of that employed throughout this thesis.

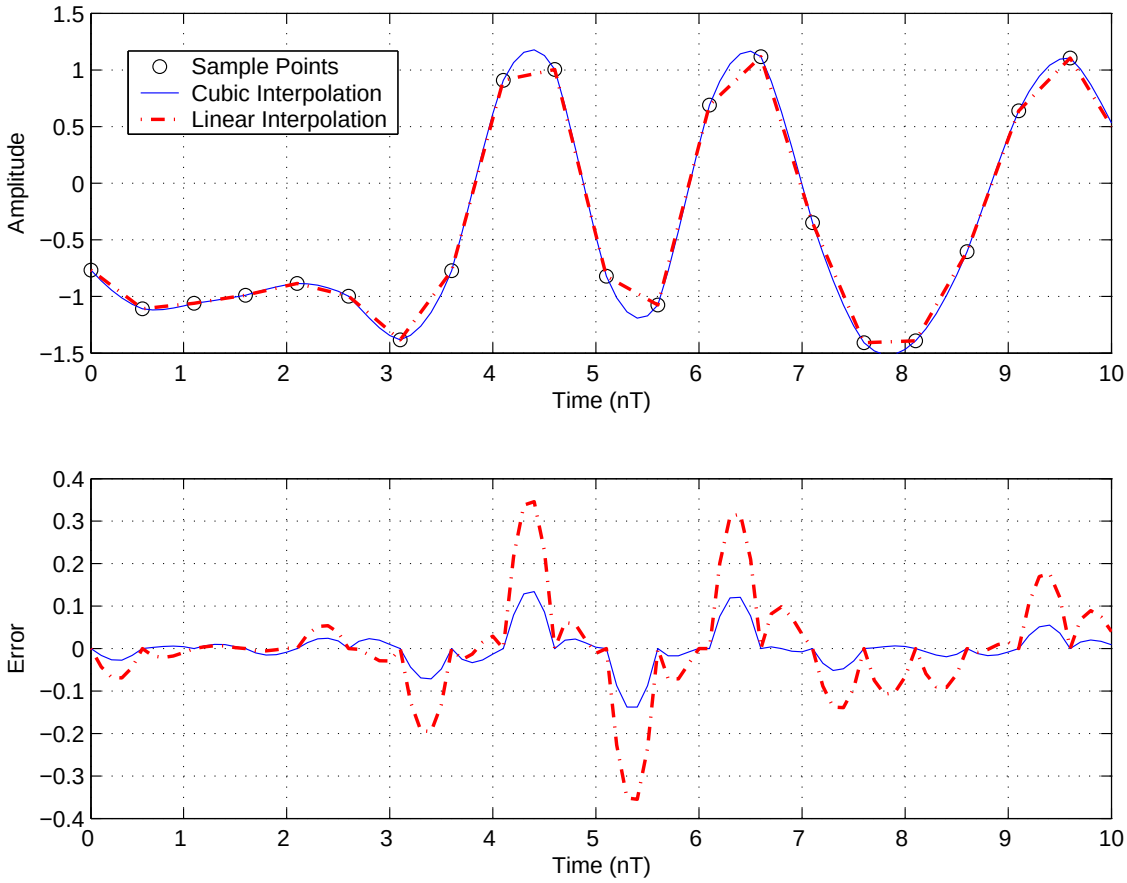

Figure 3.10: Linear v cubic interpolation comparison: interpolation of baseband signal sampled at 2 samples per symbol, for a signal matched filtered, and corrupted by AWGN at a ratio of 10db SNR. The error curves in the lower plot show the difference between each of the interpolated curves of the upper plot and the original, received analog signal.

The error variance of the cubic interpolated response is 1.9×10^{-3} and the variance of the error of the linearly interpolated response is 1.4×10^{-2}. The effect of this difference is explored in later chapters.

3.3.3 Timing Error Detection

A *Timing Error Detection* (TED) algorithm estimates the timing error (ε) for receiver timing. A TED estimates the optimal time to sample the received pulse stream. The work of [22] presented three different TEDs applicable for the type of DPLL structure illustrated in figure 3.9. These employed different methods using only information available at the receiver. A TED that estimates the timing error ε based only on symbol estimates produced by the interpolator is defined as a *Non Data Aided* (NDA) TED [22], whereas a

TED that also uses symbol information is defined as a *Decision Directed* (DD) TED.

The TED that is employed throughout much of the work of this thesis is presented in section 3.3.4 and is known as *FFML1* [8]. FFML1 uses symbol information as well as interpolator generated symbol estimates to estimate ε, and hence is a DD TED algorithm.

There are two decision directed modes of implementation for FFML1;

1. *Optimal DD mode,* where the receiver has prior knowledge of the transmitted symbols, through the use of a *training sequence*

2. *Sub-Optimal DD mode,* where there is no prior knowledge of the transmitted symbols, but estimates of transmitted symbols are produced within the DPLL to effectively replicate a training sequence. This is therefore a NDA method, but it is worth differentiating from TED algorithms that work in the absence of training symbols either estimated or known.

It is the sub-optimal implementation that is primarily investigated in the course of this thesis. Figure 3.11 illustrates the structure of the DD TED, where the loop control filter is represented by the function $LF()$ and the the variable G represents a gain block included with the loop filter function.

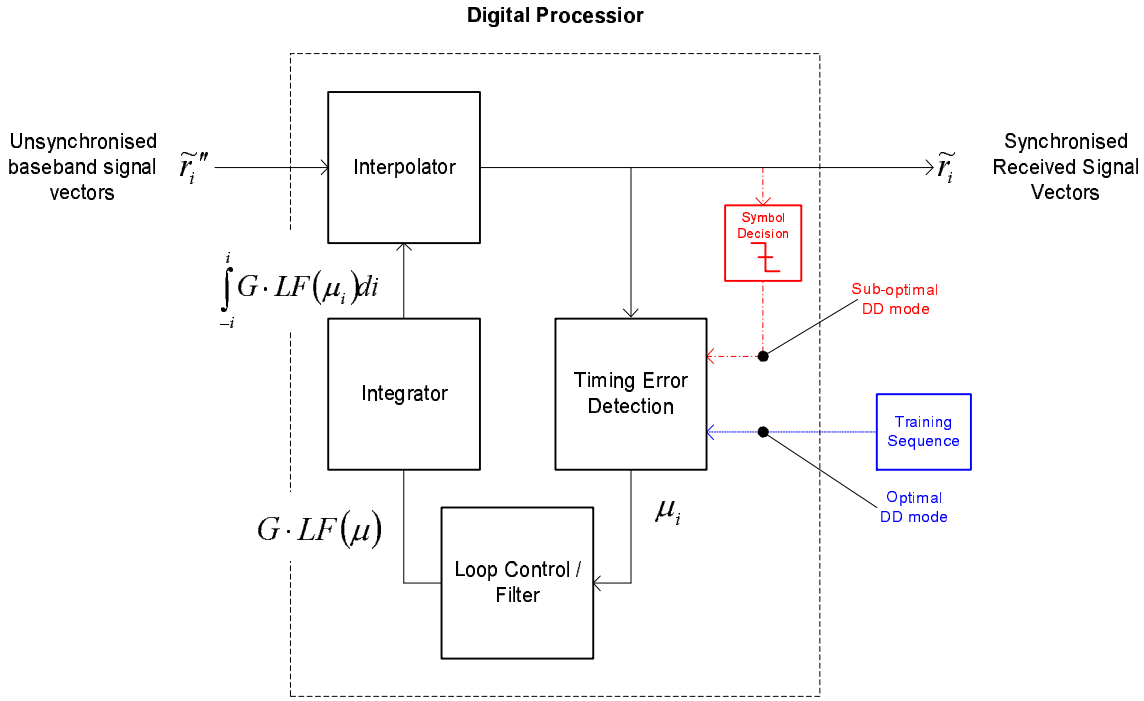

Figure 3.11: Structure of Decision Directed TED

The symbol decision made by the symbol Decision block within the schematic of figure 3.11 is only utilised by the TED of the DPLL, and is not associated with the actual symbol or data estimates produced at the digital receiver output.

3.3.4 FFML1

FFML1 is a maximum likelihood based TED algorithm [8]. FFML1 is designed as a timing recovery technique that uses samples of a received signal taken at the symbol rate. The algorithm was developed for operation in a flat fading channel, based on the Maximum Likelihood (ML) principal. This research extended the Mueller and Müller [30] family of algorithms.

For the purposes of this thesis, FFML1 is implemented as a decision directed function, in both an optimal[11] or sub-optimal form. The FFML1 function is defined by

$$\begin{aligned} \mu_i &= \left[\frac{\tilde{r}^{\star}_{m-1}\cdot\tilde{r}_{m-1}}{\tilde{a}_i\cdot F_0\cdot\tilde{a}^{\star}_i+\sigma^2_i}-1\right]\times\left[\frac{F_0}{\tilde{a}_{i-1}\cdot F_0\cdot\tilde{a}^{\star}_{i-1}+\sigma^2_i}\right]\times \\ &\quad \left[\tilde{a}^{\star}_{i-1}\cdot[\tilde{a}_i-\tilde{a}_{i-2}]+\tilde{a}_{i-1}\cdot[a_i-a_{i-1}]^{\star}\right] \end{aligned} \tag{3.12}$$

where μ_i is the timing error estimate, F_0 is the covariance of the channel fading and σ^2_i is the noise power. The index m indicates a signal with a sample rate of twice the symbol rate, whereas the index i indicates a signal with a sample rate at the symbol rate. The signal $\tilde{r}_{m-1}$ occurs simultaneously with the symbol estimate $\tilde{a}_{i-1}$.

FFML1 relies on the received signal vector and corresponding symbol estimates[12]. The function μ_i is an estimate of the timing error at that point.

3.3.4.1 Characteristics of FFML1

To demonstrate the phase detection characteristics of FFML1, a MatLab simulation was developed to produce a series of baseband complex signals, comprised of 5 symbols made up of SRRC pulses representative of the r^I_m and r^Q_m components of a QPSK signal. On each

[11] Implies a training sequence is in use.

[12] These symbol estimates can either be final estimates as would be produced by the symbol estimation block of figure 3.8 or estimates produced by the DPLL for internal DPLL use only.

of these baseband components FFML1 was calculated for the centre pulse, for timing offsets between $-0.5T$ to $0.5T$. Actual symbol information was input to FFML1, so the simulation represented the *optimal* version of FFML1. Figure 3.12 depicts the noise free result for the *mean* detection characteristic based on 10^5 symbol sequences.

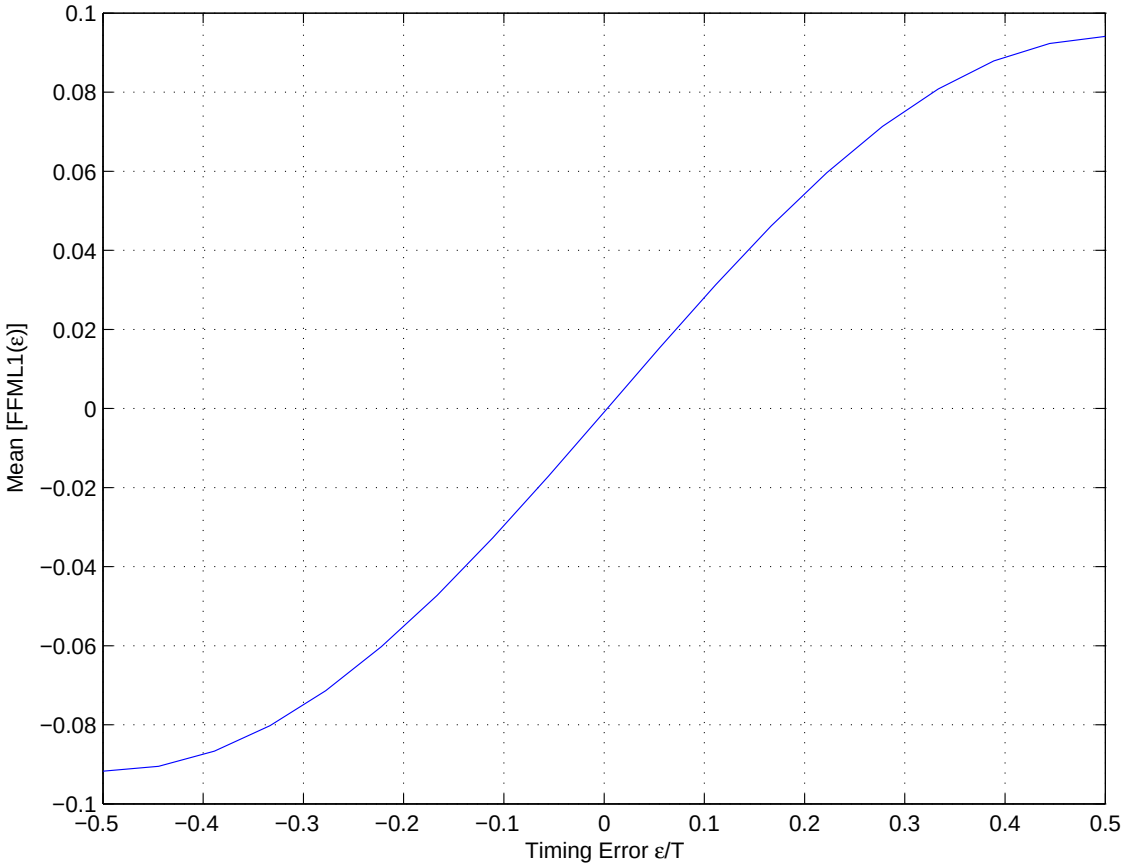

Figure 3.12: FFML1 response as a function of ε.

The so-called *s curve* of figure 3.12 closely resembles that of a sine function. It then follows that an approximation [8] for FFML1 for operation of a phase error close to zero[13] can be defined by

$$\mu = K_m \sin(\phi_\varepsilon) \tag{3.13}$$

where K_m is the gain of the TED and ϕ_ε is the phase error.

[13] Accurate to within 4 percent when ϕ_ε is less than 0.5 radians [7].

3.3.4.2 Simulated Comparison of Optimal (DD) v Sub Optimal (NDA) FFML1

To compare performance of the optimal version of FFML1 with the sub optimal version a simulation was conducted similar to that of section 3.3.4.1, but with the baseband signal corrupted by AWGN at 10dB SNR. Both optimal and sub optimal algorithms were simulated. The comparison of the two versions of FFML1 produced by the simulation is shown in figure 3.13.

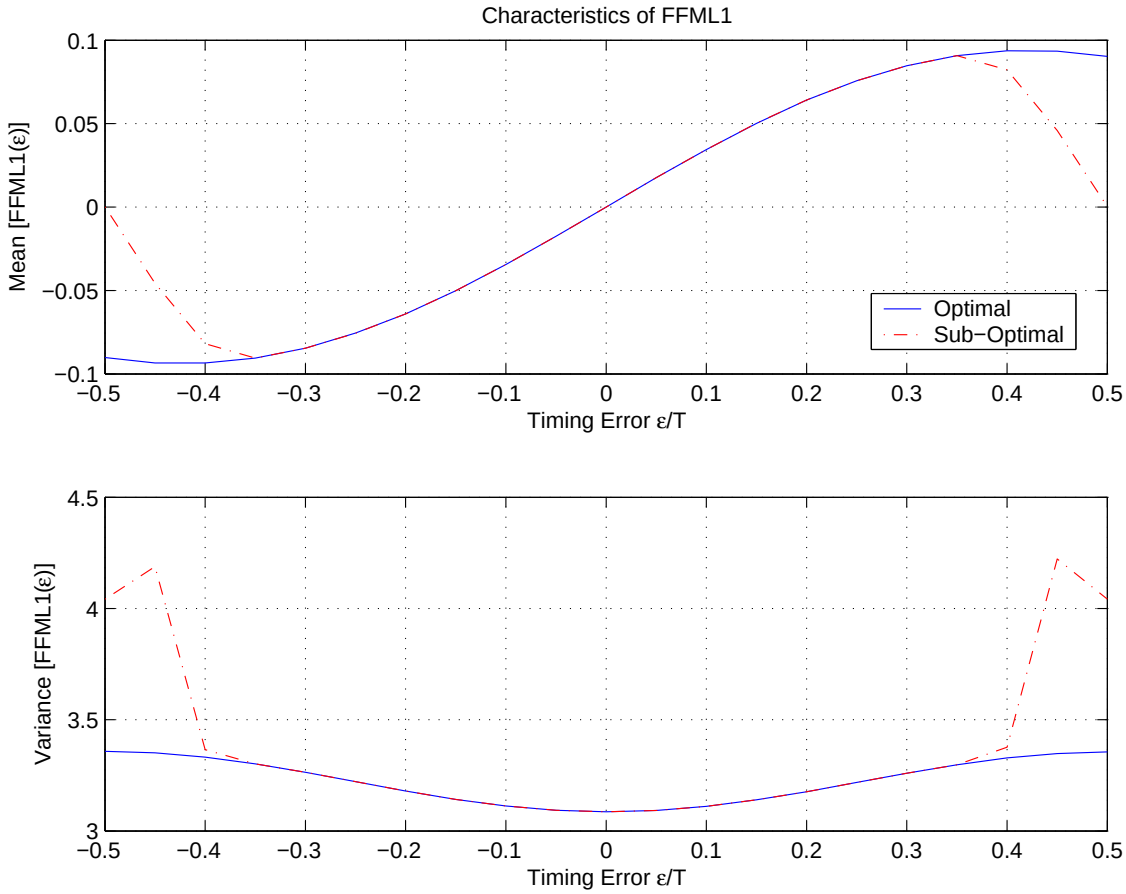

Figure 3.13: Optimal implementation of FFML1 performance compared with sub optimal version. An SRRC pulse with a rolloff of 0.35 is used corrupted by an AWGN channel with an SNR of 10dB

The key interpretation of figure 3.13 is that within an approximate phase error range of $-0.35T$ to $0.35T$ the optimal and sub optimal mean and variance characteristics of FFML1 are very similar. Where the timing error exceeds $|\varepsilon| > 0.35$ the symbol decision error rate in the sub optimal version of FFML1 begins to degrade the performance of FFML1, and the difference between the two TED implementations becomes significant.

In particular, the mean of sub optimal FFML1 reduces to zero and the ability of the TED to steer the DPLL toward the optimal timing point is also reduced to zero.

In order to explore the relationship between the error rate of the symbol decision block of the sub-optimal FFML1 implementation and the variance of FFML1 another simulation was performed, plotting the error rate of the symbol decision block against the mean of FFML1, as illustrated in figure 3.14. The errors of the symbol decision block are normalised such that they represent a *Bit Error Rate* of the transmitted data.

This simulation was performed with AWGN at 7dB SNR, in order to better show the correlation between FFML1 variance and the error rate of the symbol decision process of the sub-optimal version of FFML1.

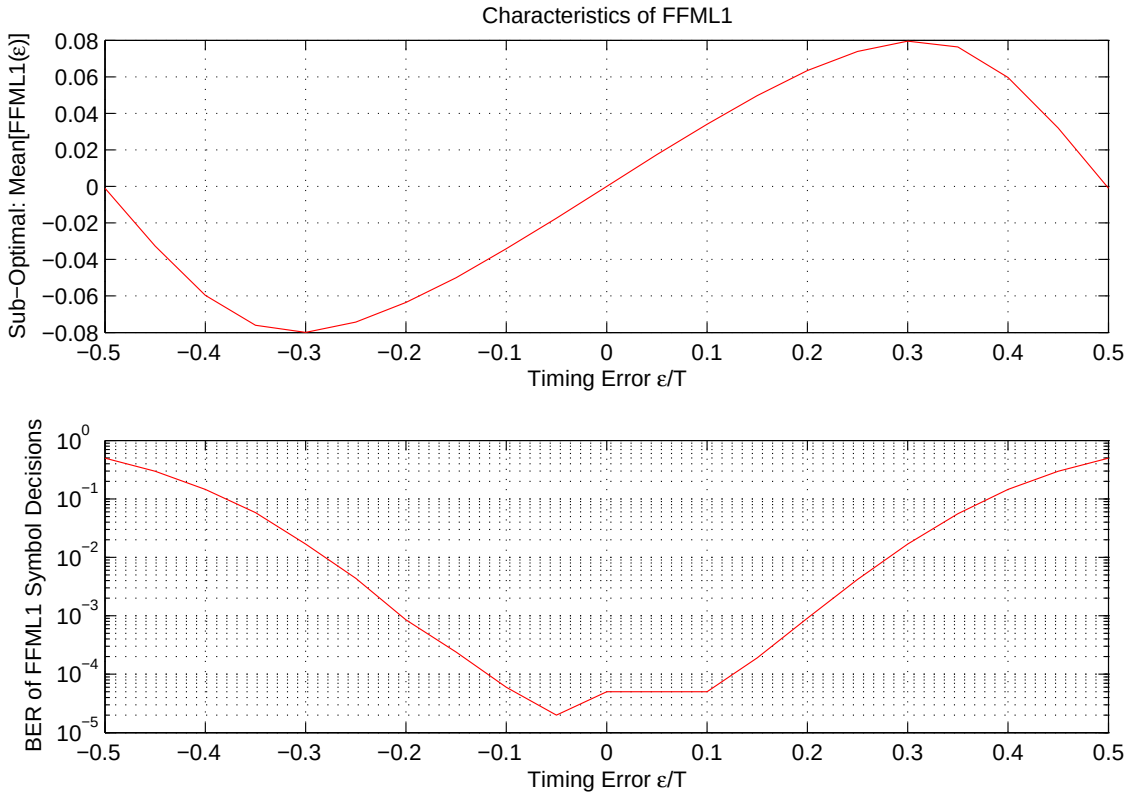

Figure 3.14: Symbol Decision error rate (BER) of the sub optimal implementation of FFML1 against Variance. An SRRC pulse with a rolloff of 0.35 is used corrupted by an AWGN channel with an SNR of 7dB.

The plots of figure 3.14 show the characteristics of sub optimal FFML1 due to errors occurring in the symbol decision block. The plot shows that as the S-curve of FFML1 begins to produce degraded steering values once the BER exceeds 10^{-2} at a timing error of $|\varepsilon| > 0.3$. *Mean[FFML1]* approaches zero as the BER approaches 0.5.

In order for FFML1 to produce useful steering values for the DPLL, a BER that would seem very high in the context of digital communications systems is in contrast acceptable for satisfactory DPLL operation. The sub-optimal version of FFML1 appears to provide a useful basis for a *blind* DPLL. The *dead zone* that occurs close to timing errors of $\pm 0.5T$ only tends to slow the acquisition time in a minority of cases.

3.3.5 Loop filter

The loop filter provides smoothing of the loop feedback signal that reduces the effect of noise and reduces the resultant variance of the DPLL timing estimate μ. Typically either a first order or second order filter is used for DPLL applications.

A first order DPLL system in phase lock, or close to phase lock, can be modeled as a linear closed loop system with a transfer function expressed [31] in the z domain by

$$H_{LF}(z) = \frac{b_0 + b_1 \cdot z^{-1}}{1 - a_1 \cdot z^{-1}} \tag{3.14}$$

where b_n's are the system *zeros* and a_n's the system *pole*s, similarly a second order DPLL closed loop system has a transfer function expressed in the z domain as

$$H_{LF}(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-1}}{1 - a_1 \cdot z^{-1} - a_2 \cdot z^{-2}} \tag{3.15}$$

where the b_n's and a_n's are system zeros and poles respectively, with the order of the loop determined by the maximum index of the a_n's.

As discussed in [8], due to feedback inherent in the FFML1 TED, a loop employing an n^{th} order filter will result in an $(n+1)^{th}$ order loop due to the pole of the TED effectively cascaded with the pole of the loop filter. Therefore for implementation of a second order DPLL, a first order filter is required.

A common principal of PLLs is that a first order loop is capable of acquiring and tracking a static phase difference, and is therefore suitable for a system where there is no frequency error between transmitter and receiver. To track the phase in a system where

there may be a frequency error present a second order loop is required. It is assumed that higher order filters and loops would be of little practical value.

The type of filter employed in the practical implementations of the DPLL followed the practice of earlier research [8, 22], a first order filter with a pole located at 0.9 and the gain implemented as a simple multiplier. The transfer function of the filter is defined by

$$H_{LF}(z) = \frac{1}{1-0.9z^{-1}} \tag{3.16}$$

3.3.6 DPLL Integration Block

The *integration* block integrates the TED calculated phase error μ over time such that the DPLL can produce a static phase offset that represents the steady state timing error. The transfer function of a single-iteration digital integrator is defined as

$$H_{VCC}(z) = \frac{T}{z-1} \tag{3.17}$$

where T is the symbol period.

3.4 FFML1 Performance Comparison with Other TEDs

This section briefly describes two other TEDs that research conducted by Clarke [22] used as a comparative basis for an evaluation of the operation performance of FFML1. The two TEDs were the *Gardiner* TED [32] and the *Amplitude Directed* TED [33]. The channel types for which the comparison was conducted were a combination of AWGN and flat fading channels, for varying levels of open loop gain. This section concludes with a summary of the conclusions of [22] as a *validation* of the use of FFML1 in this thesis.

3.4.1 Gardiner TED

The Gardiner TED [32] is a heuristic algorithm designed for a QPSK type modulation. The TED produces timing error information without symbol information[14], using either a training sequence or symbol decision within the algorithm. The algorithm takes the form

$$\varepsilon_i = r^I_{m-\frac{1}{2}} \cdot \left[r^I_m - r^I_{m-1}\right] + r^Q_{m-\frac{1}{2}} \cdot \left[r^Q_m - r^Q_{m-1}\right] \tag{3.18}$$

As indicated by the index of the received samples $r_{-m}, r_{m-\frac{1}{2}}, r_{m-1}$, the Gardiner TED uses samples of the received signal at a rate of 2 samples per symbol. Channel statistics such as the noise power are not employed, and the implementation of the TED requires only multiplication, addition and subtraction, resulting in an algorithm readily implemented on a DSP platform.

3.4.2 Amplitude Directed TED

The Amplitude Directed (AD) TED [33] is a TED designed for QPSK type modulation schemes using complex mathematics rather than simply considering the in-phase and quadrature components of the received signal as individual signals. The algorithm takes the form

$$\varepsilon_m = Real\left[sgn\left(\tilde{r}_{m-1}\right) - sgn\left(\tilde{r}_m\right) r^\star_{m-\frac{1}{2}}\right] \tag{3.19}$$

where $sgn[\cdot]$ represents the *signum* function, defined by

$$sgn[x] = \begin{cases} \frac{x}{|x|} & x \neq 0 \\ 0 & x = 0 \end{cases} \tag{3.20}$$

Like the Gardiner TED, the AD TED does not incorporate the use of symbol data, nor

[14] Hence is an non data aided (NDA) TED

the use of channel statistics. The AD TED is not as conveniently implemented in DSP as the Gardiner TED, due to the requirement for a division operation[15] in the *sgn* function.

This algorithm also operates on symbols samples at a rate of 2 samples/symbol.

3.4.3 TED Comparison

This sub-section summarises the conclusions of [22], regarding the comparison of the FFML1, Gardiner TED and AD TED structures in a DPLL.

Acquisition: FFML1 acquired the symbol timing using the least number of received symbols

Tracking: Both AD and Gardiner TEDs outperformed FFML1 in terms of variance of the TED output. It was acknowledged that an increased variance of the TED output when in tracking mode did not have a significant effect on the resultant BER of the digital receiver.

Fading: The Gardiner TED suffered the most from an increase in the channel fade rate, whereas AD and FFML1 were relatively unaffected by an increase in fade rate.

Pulse-shape: AD and Gardiner are both sensitive to the rolloff rate of the shaping filters employed, a rate of less than 100% excess bandwidth producing degraded performance. FFML1 on the other hand produced improved performance as the excess bandwidth was reduced.

Complexity: AD was the simplest of the three and FFML1 the most complex. However, as will be shown[16] FFML1 can be implemented with a reduced level of complexity, comparable to that of AD and Gardiner TEDs

[15] In as DSP implementation, a division operation is iterative, as such comes with a processing cost reducing the maximum rate at which the DPLL may process incoming data.

[16] The construction of section 4.1.5 reduces FFML1 to $\mu = D \cdot (A - B)$, with variables A, B and D defined in equation 3.12.

3.5 Baseband DPLL Model

This section describes the discrete mathematical model of a baseband DPLL. This model can be readily implemented in software as is shown in later chapters where implementation into both MatLab scripts[17] and DSP Assembly language[18] are presented. Figure 3.15 illustrates the discrete mathematical baseband DPLL Model.

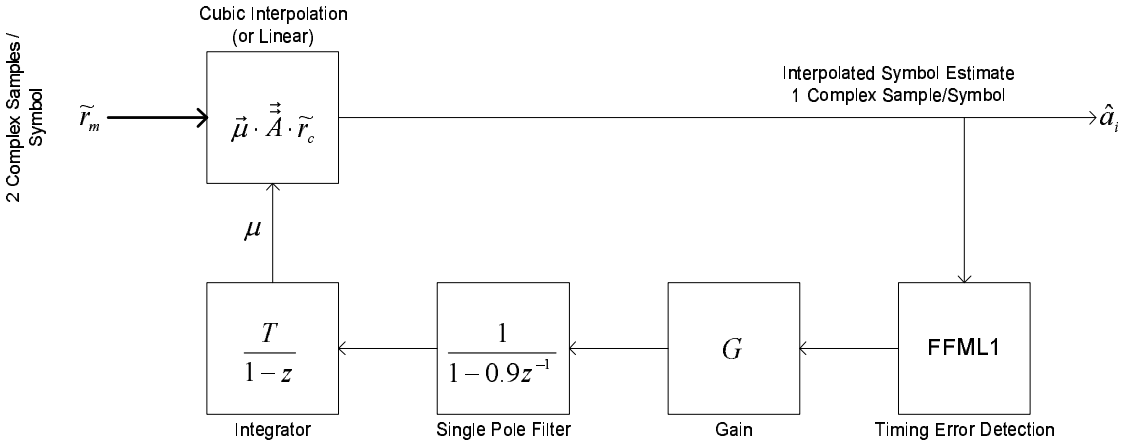

Figure 3.15: Mathematical baseband model for an implementation of an FFML1 based DPLL

The interpolation block of figure 3.15 can be implemented as either a cubic interpolator[19] (shown the figure) or as a linear interpolator[20]. The equation describing FFML1 for the TED block is presented in section 3.3.4. This model forms the basis for the software implemented in both MatLab script and DSP in chapter 4.

3.6 Summary

This chapter has introduced the concept of *timing synchronisation* as it applies in practical receiver operation. The chapter develops the basis for modeling and design of methods to recover symbol timing information.

Analog phase locked loop (PLL) theory is included to provide a foundation for theoretical concepts of PLL operation. Methods are presented for a digital receiver to use

[17] Appendix C presents the MatLab simulation scripts for an FFML1 DPLL implementation.
[18] Appendix B presents the Motorola ASM 56300 code for an FFML1 DPLL implementation.
[19] As described in section 3.3.2.2.
[20] As described in section 3.3.2.1.

analog processing as a receiver front end. In this case two levels of synchronisation are required in order to produce symbol estimate; carrier synchronisation and symbol synchronisation. Practical implementations of these analog processes are included to give the reader a feel for the level of complexity required and to illustrate the advantages available for digital methods.

Digital phase locked loop (DPLL) theory is summarised. Digital timing recovery methods are presented that operate independently of the requirement for synchronisation with the carrier (or any intermediate frequency). The received bandpass signal may be sampled using *bandpass sampling* to produce an unsynchronised complex baseband digital signal, thus reducing receiver complexity. In the case of acquiring symbol timing, the sampling clock is free running, not requiring any control or adjustment. Symbol timing recovery is achieved independent of carrier phase recovery. Synchronisation is achieved by the DPLL, made up of several functional blocks, the interpolator, timing error detector, filter and integrator. An *interpolation* filter allows the receiver to estimate the baseband signal (a symbol estimate) at any given point in time.

The *timing error detector* (TED) estimates the timing error between the current sample timing and the transmitted symbol timing and provides the error signal to drive the DPLL. Three methods for implementation of a TED are presented. FFML1 TED [8] was selected for implementation, due to superior acquisition performance and operation in the fading channel.

Chapter 4

DPLL Implementation in DSP

This chapter presents the software design and DPLL implementation on a DSP platform. The DSP environment used is the Motorola 56321 evaluation board and the software is implemented in Motorola 56000 ASM assembly language.

In the first section some practical considerations for DSP implementation are explored. The second section breaks the DPLL software design into its building blocks, then details the design of each block. The third section presents the results of the DSP hardware running the DPLL code in an off-line mode with simulated received signal samples in order to accurately assess performance of the DPLL under simulated channel conditions.

4.1 DPLL Software Design

This section describes the DSP implementation of the DPLL, and how a simulated *received signal* is created by a MatLab script with specified noise and fading corruption to measure DPLL performance.

The implementation of this section utilises the decision directed variant of the DPLL, eliminating the need to incorporate a training sequence in the simulation. This served to both simplify the configuration and demonstrate the ability of the DPLL to operate blind.

4.1.1 Design Overview

Figure 4.1 is an overall illustration of the functional software blocks of the DPLL implementation.

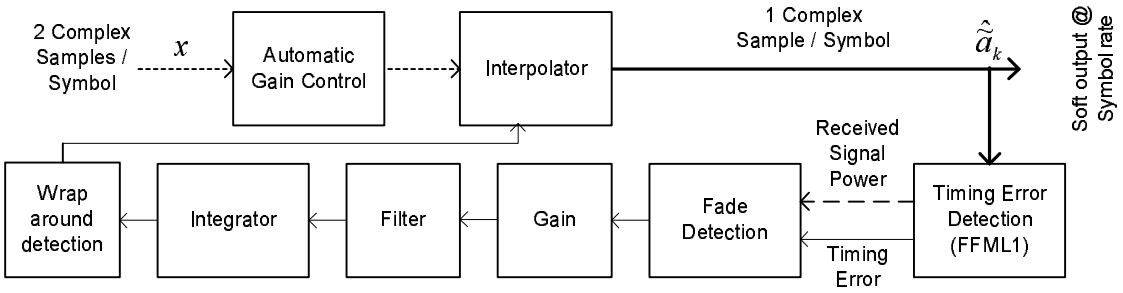

Figure 4.1: DPLL Software Design Overview

The Automatic Gain Control (AGC) block is additional to the baseband model used for the MatLab based simulations described in section 3.5. The Interpolator, TED, gain, filer and integrator blocks are equivalent to those of section 3.5, whereas the fade detection and wrap around detection blocks are added to compensate for the effects of real-time operation. The interpolator is effectively a multi-rate digital filter, decimating the input data from a rate of 2 samples per symbol to 1 sample per symbol.

4.1.2 Automatic Gain Control

The Automatic Gain Control (AGC) is implemented as a Moving Average (MA) filter utilising the co-processor of the 56321 chip, namely the Enhanced Filter Co Processor (EFCOP). The AGC operates in three parts as illustrated in figure 4.2.

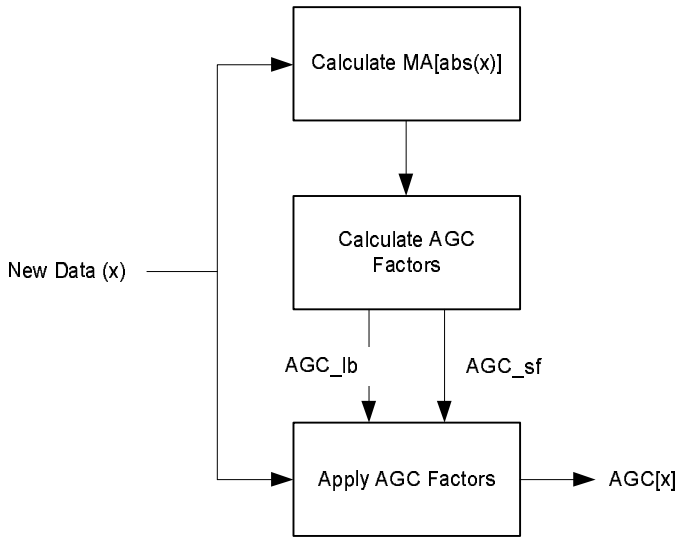

Figure 4.2: AGC Design Overview

The first part pushes each incoming sample value *x* (where *x* consists of consecutive complex components: real(sample), imag(sample), real(sample),) onto the front end of an FIR filter implemented using the EFCOP to calculate a MA of abs(x). The output of the EFCOP is then decimated by 16:1 such that for every 16 inputs, one output value is placed on the output buffer and an indication flag called the Filter Data Output Buffer Full (FDOBF) flag is then set.

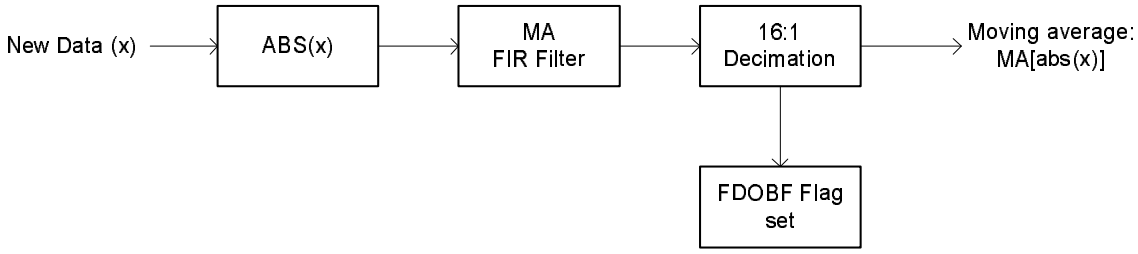

Figure 4.3: AGC, First Part

The FDOBF flag is tested once per DPLL loop cycle. When it is set the second part of the AGC is then called calculating values for bitwise normalisation and scale factor.

The second part of the AGC function is a routine that first shifts the accumulator left one bit (effectively multiplying by a factor of two) such that the count of the leading bits will be one less than the actual number so as to leave a single bit pad. The single bit padding allows for the AGC function to be applied to incoming data of up to twice the

nominal input value before an overflow occurs. The accumulator is then *normalised*[1] to the MSB-1 of the leading bits as illustrated in figure 4.4.

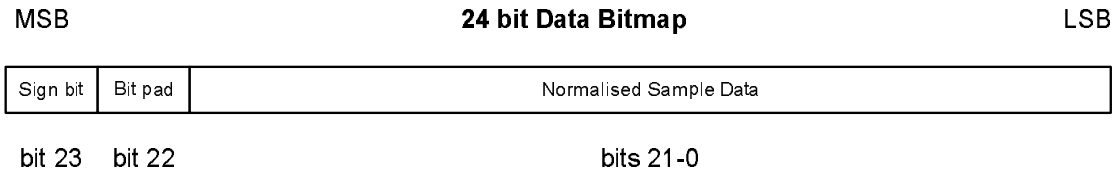

Figure 4.4: Bit map of Normalised Sample Data

The scale factor AGC_sf is calculated by division, such that

$$Norm\,(MA\,[abs(x)]) \times AGC\_sf \quad = \quad |Symbol| \tag{4.1}$$

where $|Symbol|$ is the nominal amplitude of each symbol

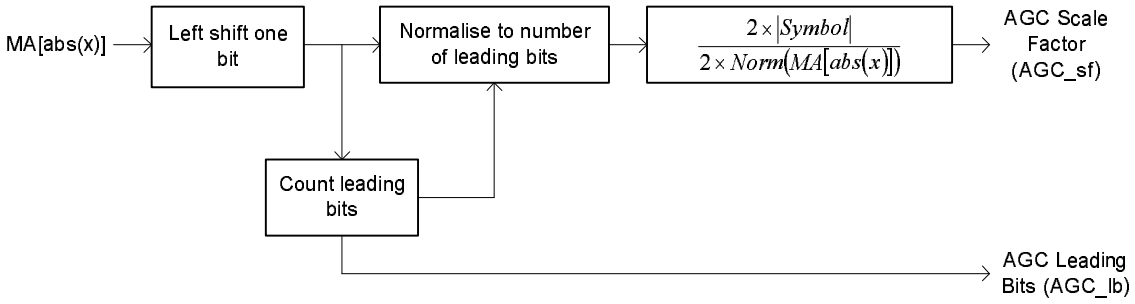

Figure 4.5: AGC Second Part

Division as implemented in a DSP56321 chip is inherently inefficient, as it requires 24 iterations (for a 24 bit number) to complete. This is mitigated by the decimation function (figure 4.3) requiring that the second part of the AGC function only be calculated once every 16 loop cycles.

Once values for leading bits and scale factor have been calculated then the third part of the AGC operation is to apply these values to each incoming sample value, as illustrated in figure 4.6.

[1] *Normalisation* in the context of 2's complement mathematics is a process of left-shifting the binary bits such that the most significant bit (MSB) occurs at a pre-determined bit number. This left shift is an *arithmetic shift*, such that the sign bit of the original data is retained.

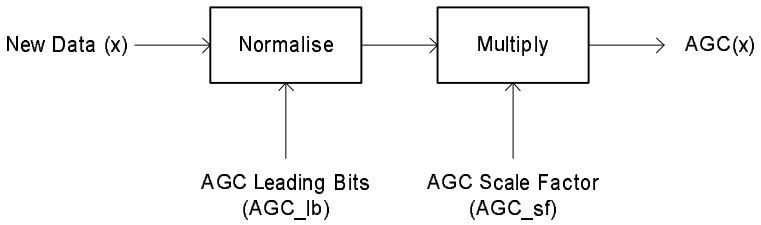

Figure 4.6: AGC Third Part

The gain adjusted input samples x_n as modeled by figure 4.2 can the be written as the complex received signal samples $\tilde{r}_m$

$$\tilde{r}_m = AGC[x_{2m}] + j \cdot AGC[x_{2m+1}] \tag{4.2}$$

where the subscript index m implies a sampling rate of two times the symbol rate.

4.1.3 Cubic Interpolator implementation

Although the matrix multiplication for the implementation of cubic interpolation seemingly requires a significant amount of processing, in all 24 Multiply Accumulate (MAC) operations and 48 data moves the DSP56321 chipset lends itself well to an efficient implementation. This matrix multiplication is implemented by means of a series of ring buffers acting as data sources supplying consecutive values to MAC operations that occur in parallel with data moves and memory pointer incrementation. Each of these MAC instructions occur over a period of two clock cycles.

The matrix operation is represented as

$$\hat{\tilde{a}}(\tilde{r}, \mu) = \vec{\mu} \cdot \vec{\vec{A}} \cdot \tilde{\vec{r}} \tag{4.3}$$

where $\hat{\tilde{a}}(\tilde{r}, \mu)$ is the interpolated (estimated) symbol and $\vec{\mu}$, $\vec{\vec{A}}$ and $\tilde{\vec{r}}$ are vectors constructed as described in section 3.3.2.2.

This matrix operation is implemented with 48 DSP instructions as the process illustrated in figure 4.7.

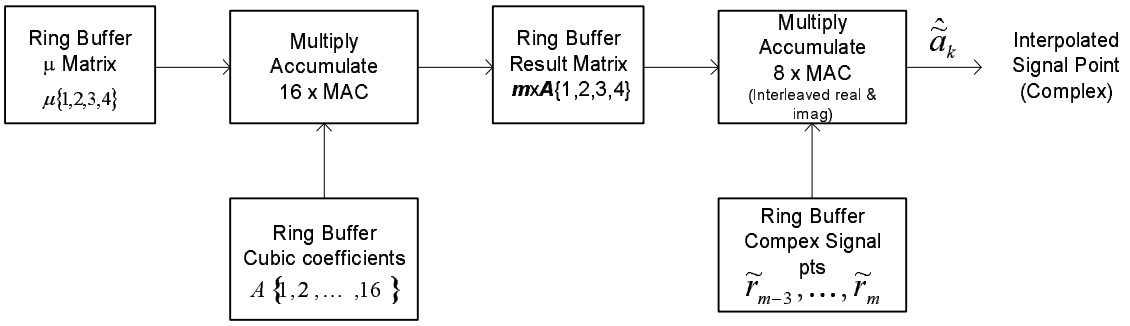

Figure 4.7: Cubic Interpolation DSP Implementation

where the product of the cubic interpolator is the symbol estimate $\hat{\tilde{a}}_k$, the subscript index of k implies sampling at the symbol rate. The received signal sample $\tilde{r}_m$ subscript index of m implies sampling at twice the symbol rate.

4.1.4 Linear Interpolation

Linear interpolation is implemented using straightforward addition, subtraction and multiplication to calculate a 2.5 sample / symbol interpolation. Linear interpolation as illustrated in figure 4.8 can be implemented with 15 DSP instructions.

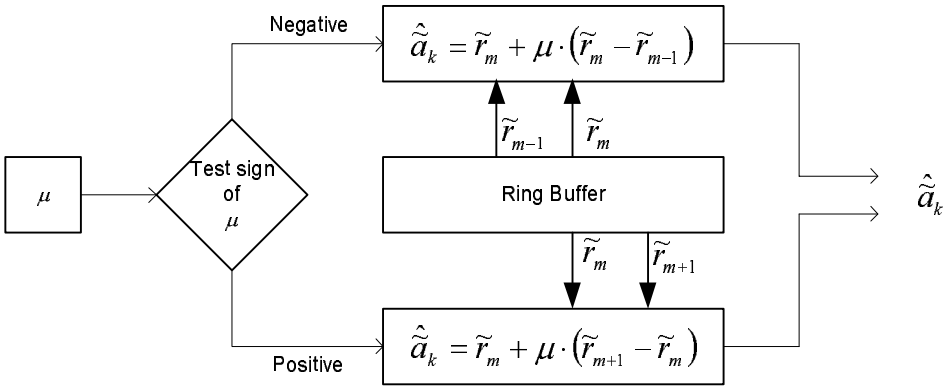

Figure 4.8: Linear Interpolation DSP Interpolation

where the subscripts of $\hat{\tilde{a}}_k$ and $\tilde{r}_m$ have the same implications as to the signal rate as in the previous subsection.

4.1.5 FFML1 Implementation

The original FFML1 equation (reproduced from section 3.3.4 for convenience) has the form

$$\begin{aligned}\mu &= \left[\frac{\tilde{r}^{\star}_{m-1}\cdot\tilde{r}_{m-1}}{\tilde{a}_{k-1}\cdot F_0\cdot\tilde{a}^{\star}_{k-1}+\sigma_n^2}-1\right]\times\left[\frac{F_0}{\tilde{a}_{k-1}\cdot F_0\cdot\tilde{a}^{\star}_{k-1}+\sigma_n^2}\right]\times \\ &\quad \left[\tilde{a}^{\star}_{k-1}\cdot[\tilde{a}_k-\tilde{a}_{k-2}]+\tilde{a}_{k-1}\cdot[\tilde{a}_k-\tilde{a}_{k-1}]^{\star}\right] \end{aligned} \tag{4.4}$$

Making the substitutions

$$\begin{aligned} A &= \tilde{r}^{\star}_{m-1}\cdot\tilde{r}_{m-1} \\ B &= \tilde{a}_{k-1}\cdot F_0\cdot\tilde{a}^{\star}_{k-1}+\sigma_n^2 \\ C &= F_0 \\ D &= \tilde{a}^{\star}_{k-1}\cdot[\tilde{a}_k-\tilde{a}_{k-2}]+\tilde{a}_{k-1}\cdot[\tilde{a}_k-\tilde{a}_{k-2}]^{\star} \end{aligned} \tag{4.5}$$

equation 4.4 then becomes

$$\mu = \frac{CD(A-B)}{B^2} \tag{4.6}$$

For D-QPSK modulation we can assume a (nominal) constant power constellation. When this is coupled with static values for the normalised fade rate F_0 and AWGN σ_n^2, then variables B and C in equation 4.5 become constant. It then follows that the values for $\frac{1}{B^2}$ and C in equation 4.6 are also constant, and can be absorbed into the loop gain as static multipliers. As a consequence, the requirement for a division operation is eliminated.

The simplified, constant power, constant channel form of FFML1 is then

$$\mu = D\cdot(A-B) \tag{4.7}$$

where B is a pre-computed value given by equation 4.5.

Following some algebraic manipulation the variables parts A and D of FFML1 can be implemented within the DSP as functions of I and Q as

$$\begin{aligned} A &= real\,(\tilde{r}_m)^2 + imag\,(\tilde{r}_m)^2 \\ D &= real\,(\tilde{a}_{k-1}) \times [real\,(\tilde{a}_k) - real\,(\tilde{a}_{k-2})] + \\ &\quad imag\,(\tilde{a}_{k-1}) \times [imag\,(\tilde{a}_k) - imag\,(\tilde{a}_{k-2})] \end{aligned} \tag{4.8}$$

FFML1 can be implemented into DSP based on equations of 4.8 as illustrated in figure 4.9

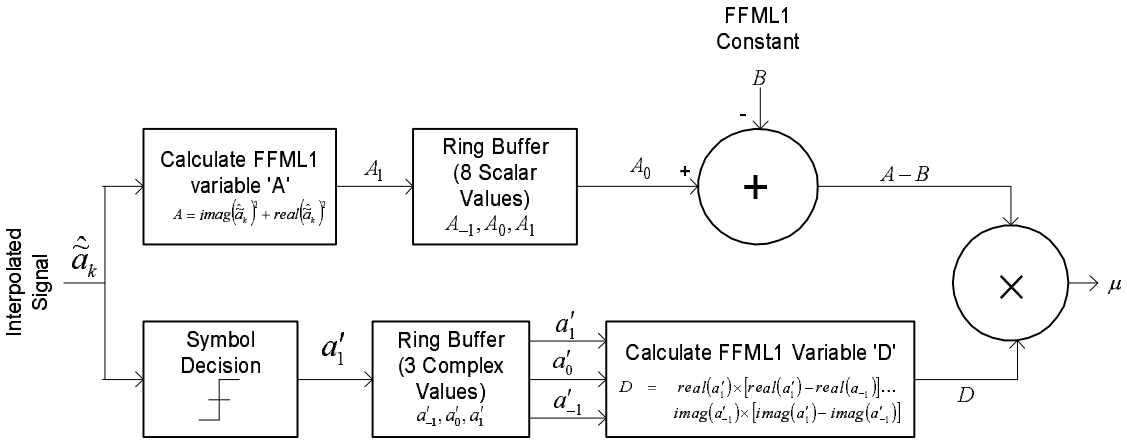

Figure 4.9: FFML1 Operation

The symbol decision block of figure 4.9 is a means to implement the decision directed variant of the DPLL; the resultant symbols produced by this decision block are not used for a hard-output from the DPLL, but for an immediately derived variable for the calculation of FFML1.

4.1.6 Fade Detection

The fade detection block is implemented as a short-term MA filter that averages the power of the interpolated symbol estimate, calculated as variable A in equation 4.8.

When the MA calculates an average signal power that is below a threshold level, then loop phase adjustment is frozen by setting the loop gain to zero.

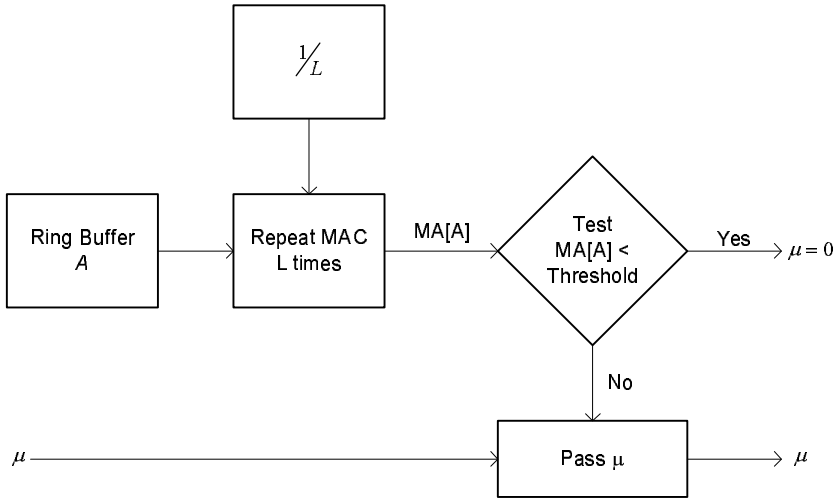

Figure 4.10: Fade Detection

4.1.7 Loop Gain and Filter Blocks

The loop gain block is implemented as a simple multiplication and the single pole IIR filter as a Multiply Accumulate (MAC) instruction. The model for implementation of the gain and filter is illustrated in figure 4.11.

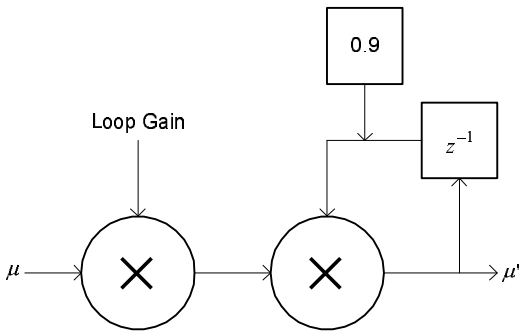

Figure 4.11: Loop Gain and Filter Processes

Mathematically the implementation of the loop gain and filtering operations are defined by

$$\mu' = \frac{\mu \cdot LG}{1 - 0.9z^{-1}} \tag{4.9}$$

4.1.8 Integration

In [22] a model for a discrete time integrator was developed from a bilinear transform, where a pair of difference equations were developed as

$$\begin{aligned} y_n &= \frac{1}{2f_s} \cdot [w_n + w_{n-1}] \\ w_n &= x_n + w_{n-1} \end{aligned} \tag{4.10}$$

Removing constant multipliers and dividers, which can be absorbed into the loop gain multiplier, these difference equations can be reduced to the series of operations,

$$\begin{aligned} I_1 &= \mu' + I_2 \\ \mu'' &= I_1 + \mu' \\ I_2 &= I_1 \end{aligned} \tag{4.11}$$

where μ' is the pre-integration value, μ'' is the post integration value and I_1 and I_2 are buffer variables for use in the integration process.

The model for implementation of discrete time integration is illustrated in figure 4.12.

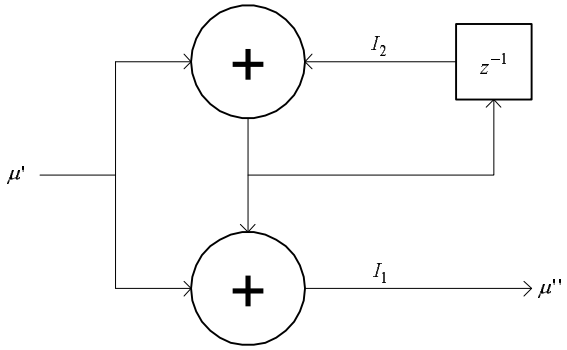

Figure 4.12: Discrete-Time Integrator Model

The implementation of this integrator in ASM code is efficiently implemented with five instructions.

4.1.9 Phase Wrap-Around Detection

A *phase wrap-around* occurs when the calculated phase error μ exceeds the maximum number possible in the DSP56321 chipset. The DSP56321 uses *fractional numbers*, and the only numbers that can be represented are those that lie between 1 and +1. When two numbers are added or subtracted such that the result lies outside that range, then a wrap around has occurred.

Detection of phase wrap-around is achieved by testing the arithmetic overflow bit in the DSPs status register (SR). The overflow bit (V) is a *sticky bit*. When it is set by an arithmetic operation, the V bit will remain set until cleared by a specific command to do so. For the purposes of wrap-around detection, the V bit is cleared prior to integration taking place. Following the integration process, if overflow has occurred then the current sign of μ is compared with the previous sign to determine if the wrap-around has occurred in a negative or a positive direction. Given the direction of a wrap-around occurrence, the time index of the incoming samples is effectively incremented or decremented such that the received symbol series will remain in synchronisation.

The adjustment of the index of incoming samples is achieved by reading in one new complex data sample (instead of two) in the case of a negative-wrap or by reading in three new samples in case of a positive wrap-around.

4.2 MatLab Simulated Performance

This section presents a DPLL implementation in MatLab based on the software design presented in section 4.1. Here we employ double precision floating point arithmetic for a high degree of accuracy. That presents an *idealistic* response for the DPLL free of quantisation noise. The AGC function is not employed as the simulated received signal is created with a predetermined average.

This subsection is not intended to be an exhaustive analysis of the DPLL, but to illustrate DPLL performance and to provide a basis for comparison with the DSP implemented DPLL.

The DPLL implmentation of this section is the decision directed variant.

4.2.1 Simulation Procedure

This procedure is encapsulated within a single MatLab script.

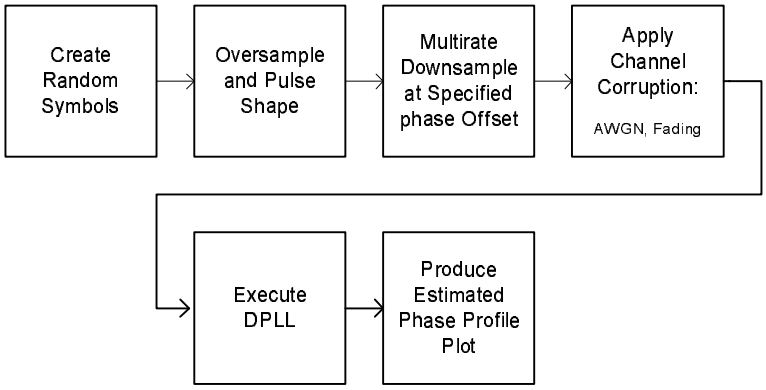

Figure 4.13: MatLab implemented DPLL simulation

Figure 4.13 illustrates the MatLab implemented DPLL simulation process, designed to create simulated received data, execute the DPLL and produce a plot of the estimated phase against the symbol number.

4.2.2 DPLL Performance Metrics

To evaluate how well the DPLL implementation was working, two main metrics were utilized; namely the mean and variance of the estimated phase timing error. A plot of mean phase error against received symbol number shows the rate at which the DPLL acquires the phase offset of the received signal. The variance of the estimated phase when the DPLL is in a phase-locked state is a measure of the DPLLs ability to deal with channel corruption. The DPLL employed in this chapter uses the decision directed implementation, to track a known phase error, which in each case is 0.6.

4.2.3 Rate of Phase Acquisition

The rate of phase acquisition varies with the loop gain of the DPLL. The following plots show the phase acquisition profile for both the linear interpolator and cubic interpolator versions of the DPLL at 0.01, 0.02 and 0.05 loop gain values.

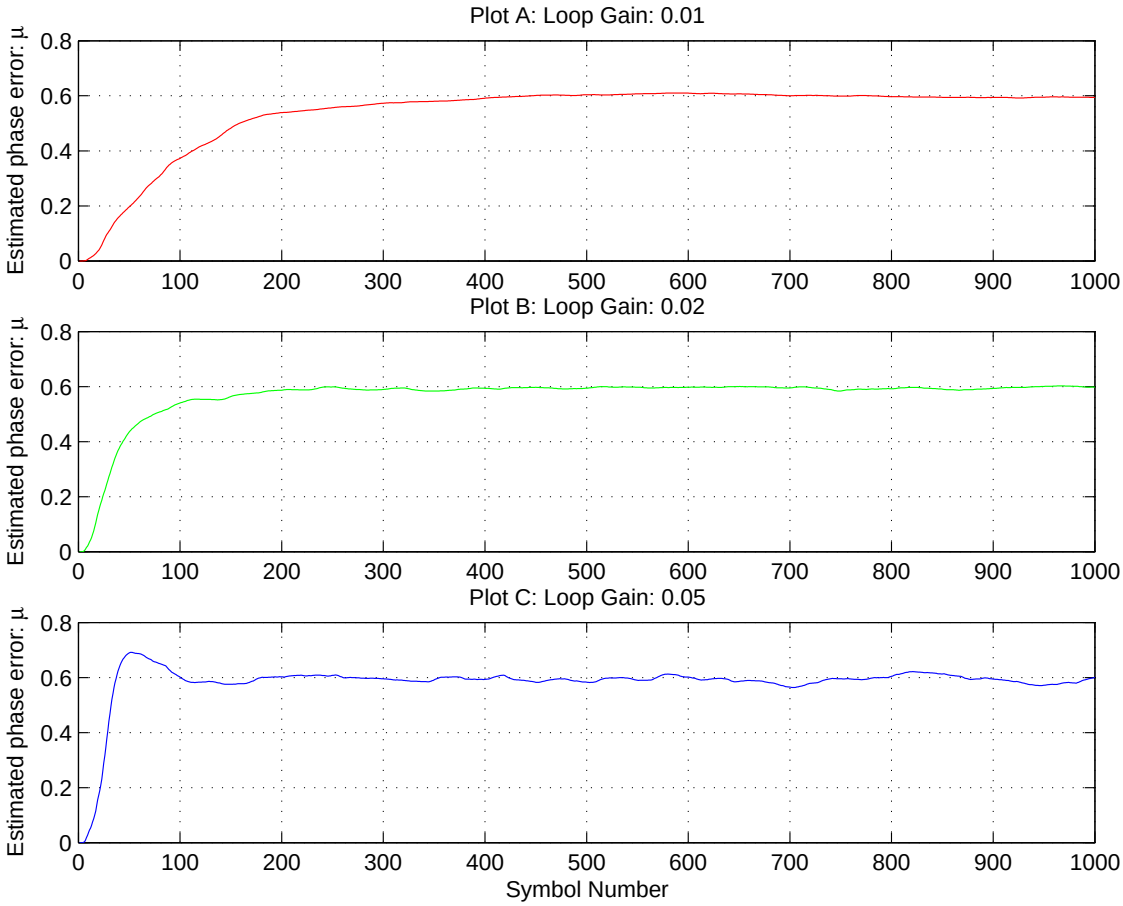

Figure 4.14: Phase acquisition profile for the linear interpolator version of the DPLL in a noise-free channel tracking a known phase error of 0.6

Plot A of figure 4.14 with loop gain of 0.01 fully acquires the phase offset in approximately 500 received symbols. Plot B approximately halves that, requiring around 250 symbols, while loop gain of 0.05 overshoots and properly acquires the phase after approximately 100 symbols.

Figure 4.15 represents profiles for the same loop gain as figure 4.14, but using the cubic interpolator version of the DPLL.

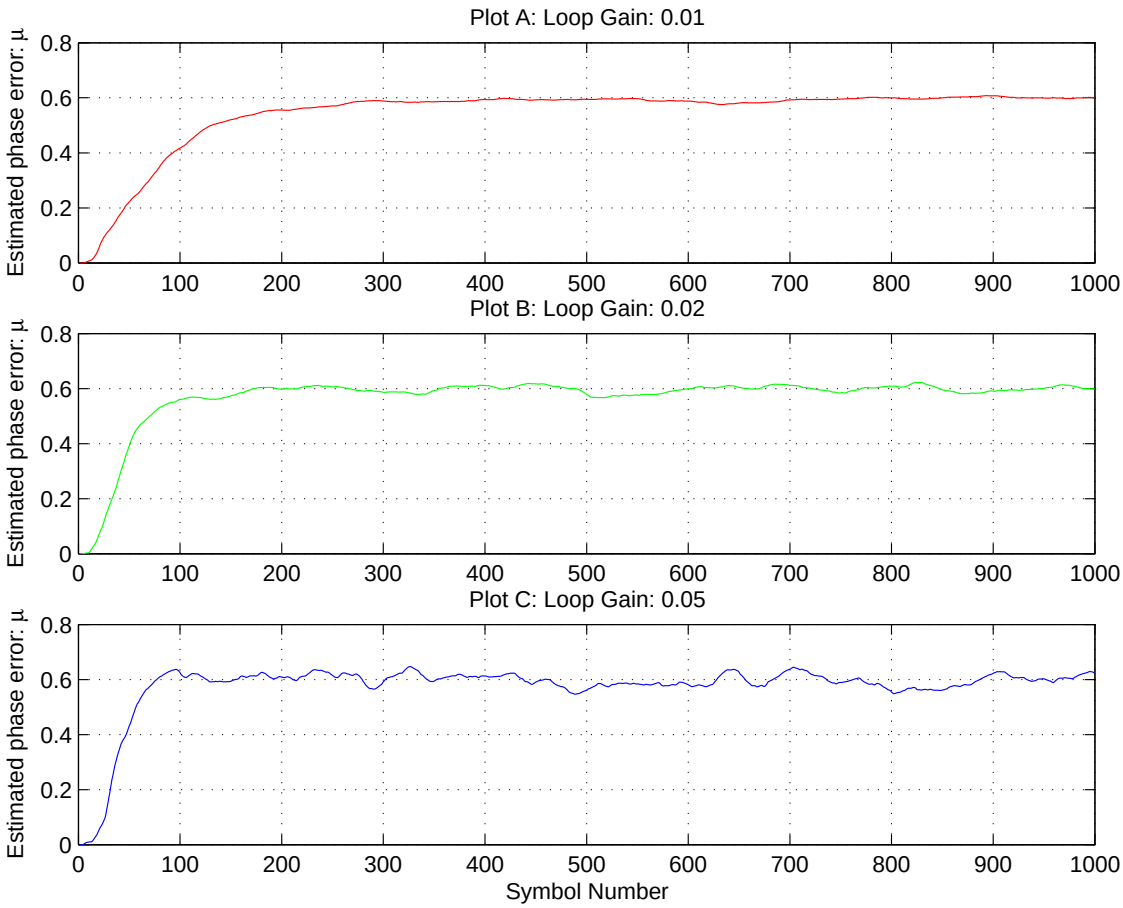

Figure 4.15: Phase acquisition profile for the cubic interpolator version of the DPLL in a noise-free channel tracking a known phase error of 0.6

Plot A of 4.15 illustrates a DPLL with a loop gain of 0.01 acquiring phase lock after receiving about 300 symbols, plot B illustrating DPLL phase lock performance for loop gain of 0.02 takes 150 symbols and Plot C acquires phase lock with 100 symbols.

4.2.4 Performance in White Gaussian Noise Corrupted Channel

To assess performance when receiving a signal corrupted by AWGN simulated received data was processed by the DPLL to produce plot of performance in varying SNR level. Results are plotted in figure 4.16.

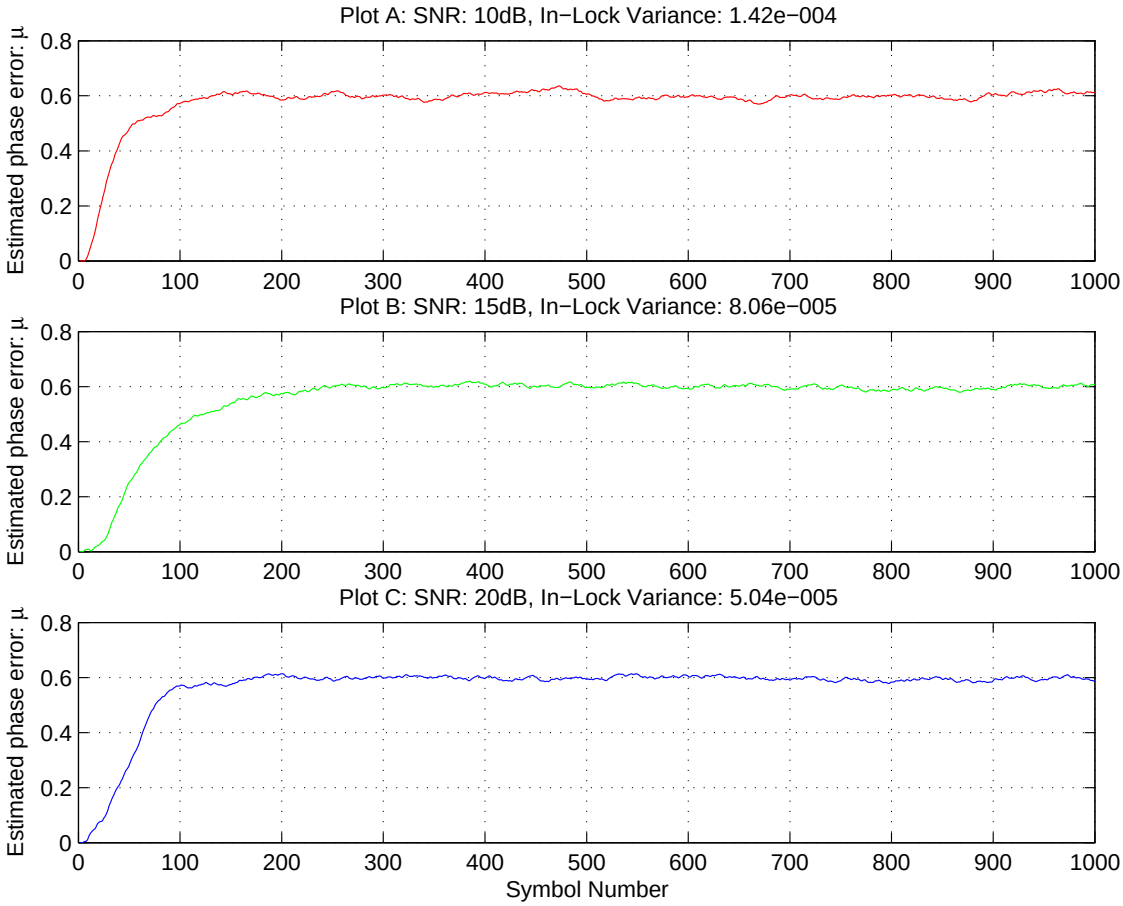

Figure 4.16: Profile of phase estimate for a DPLL implemented with a linear interpolator and a loop gain of 0.02 for varying SNR levels

The plots of figure 4.16 show the phase of the signal being acquired and tracked after about 200 symbols. The three plots look similar in profile. The effect of increasing SNR[2] from plot A to plot C is seen to reduce the variance of the estimated phase when the DPLL is in phase lock.

4.2.5 Performance in a Flat Fading Channel

The following two plots show the estimated phase error as the second order DPLL acquires and tracks a signal that has been corrupted by noise in a flat fading channel.

[2]The SNR is determined by the process described in section 2.5.1

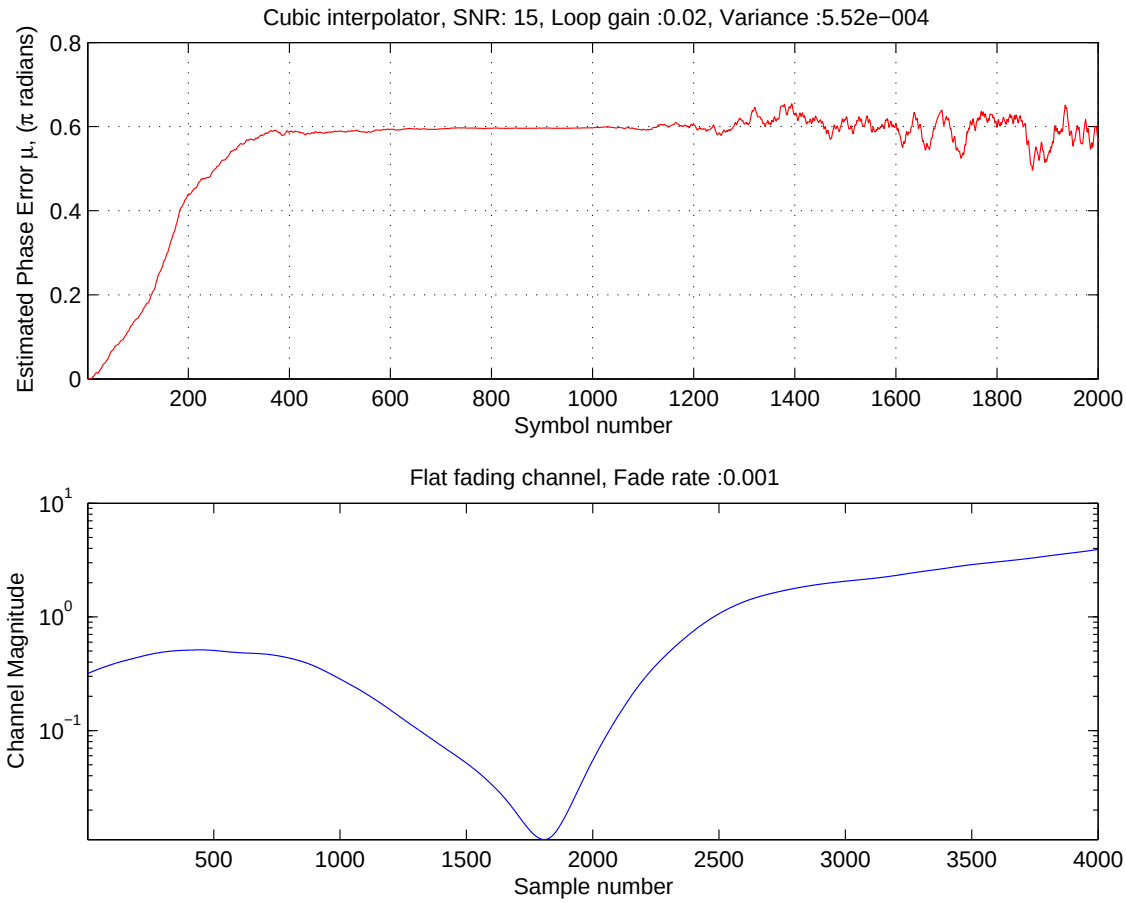

Figure 4.17: MatLab simulated DPLL with received data corrupted by a flat fading channel with normalised fade rate of 0.001 and AWN of 15dB SNR.

Figure 4.17 illustrates a channel that is varying at a relatively slow rate with a normalised fade rate of 0.001 and an AWGN of 15dB SNR. The phase is acquired after reception of about 400 symbols, which is significantly slower than a channel with the same AWGN and no fading as depicted in figure 4.16. The other characteristic that is evident is the increasing variance in the estimated phase error as the channel magnitude increases above unity, after 2500 received symbols.

The next figure illustrates performance as the fading rate is increased to a normalised fade rate of 0.003.

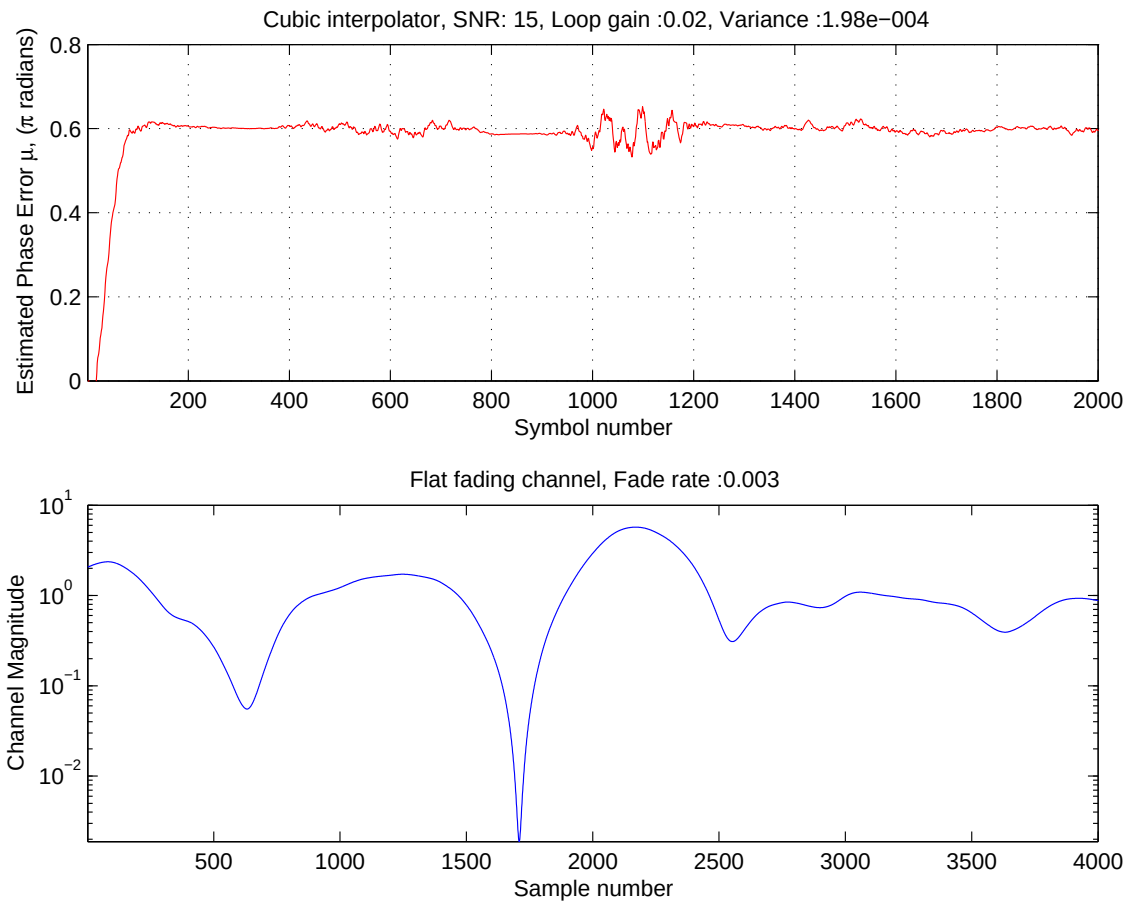

Figure 4.18: MatLab simulated DPLL with received data corrupted by a flat fading channel with normalised fade rate of 0.003 and AWN of 15dB SNR.

In figure 4.18 the phase is acquired after 100 symbols have been received. This acquisition occurs faster than was observed in figure 4.17 where 400 symbols were required. The faster acquisition is attributed to the channel state of the latter having a magnitude close to unity. The phase is tracked with a small variance evident until the point at which about 1100 symbols have been received, where the variance of the estimated phase appears to spike, with an up-fade. As with the observation made with regard to figure 4.17, this increase in estimated phase variance occurs at a point where the channel magnitude has peaked.

4.3 DSP Simulated Performance

This section presents measurements of DSP implemented DPLL performance using a simulated received signal created using MatLab scripts. The DPLL software was executed

on the Motorola 56321 DSP board, and retrieved received data from DSP memory rather than reading from the output buffer of the SASRATS receiver hardware. A second order DPLL loop is again employed throughout this section. The received signals processed by the DSP implemented DPLL are created by a MatLab script.

The DPLL implmentation of this section is the decision directed variant.

4.3.1 Simulation Procedure

The simulated received signal is created with a pre-configured level of attenuation. AWGN and / or flat fading is applied to D-QPSK encoded symbols pulse shaped with an SRRC pulse with a roll-off of 0.35 sampled at a rate of 2 samples per symbol. The process is illustrated in Figure 4.19.

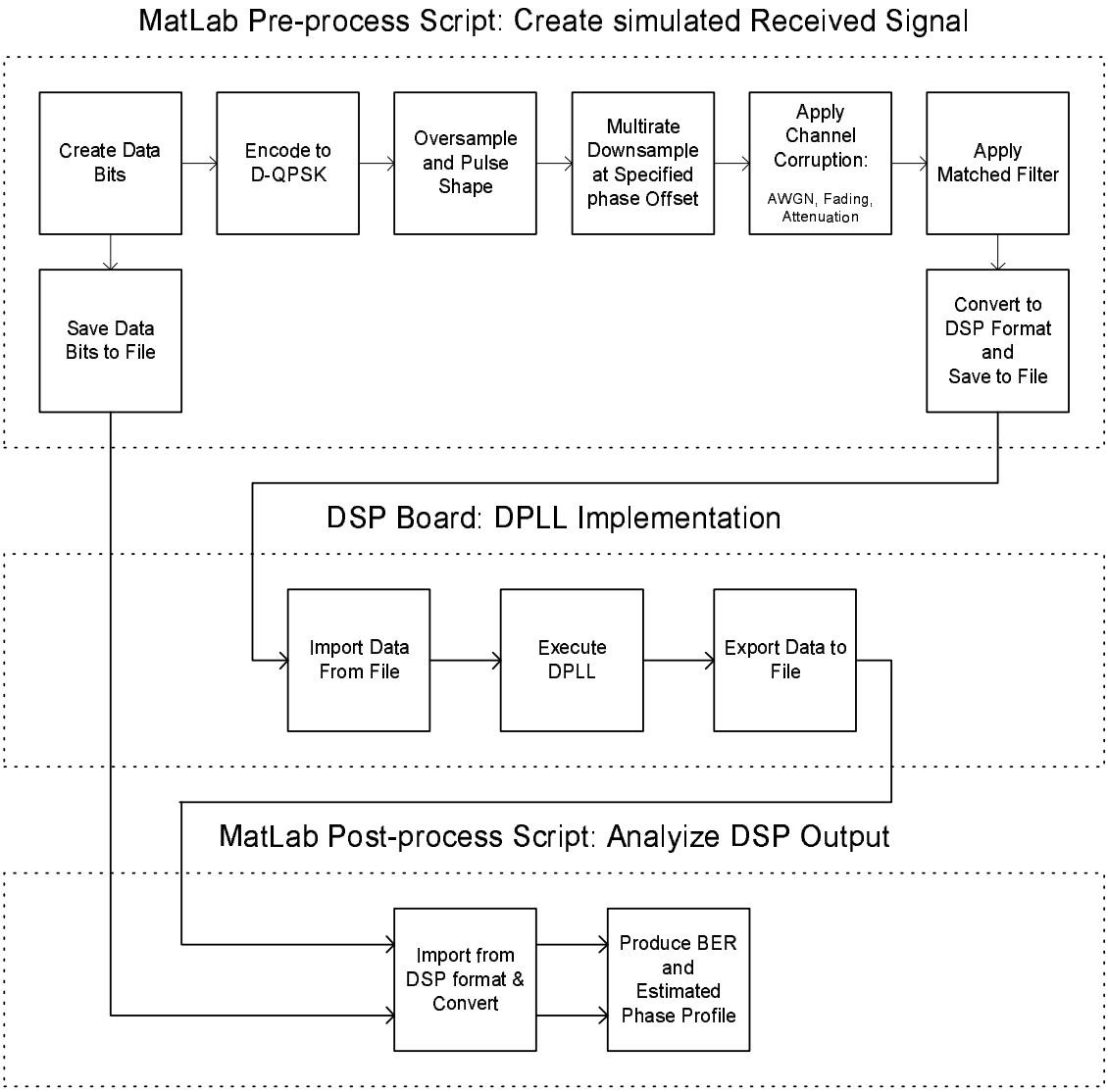

Figure 4.19: MatLab Script: Creation of a Simulated Received Signal

The simulated received signal is then dumped to the memory of the DSP board. The DPLL software is executed for a fixed number of loop cycles, and the resultant symbol estimates and calculated phase errors are recorded to DSP memory. Matlab post-processing is used to import the output from the DSP, convert and display plots of data, using pre-saved data bits to calculate the BER performance.

4.3.2 Rate of Phase Acquisition

To assess the rate of phase acquisition, a simulated received signal was created with a phase offset of $0.25 \times \pi$ radians, at 30dB SNR. The profile of the estimated phase offset,

μ, is illustrated in figures 4.20 and 4.21.

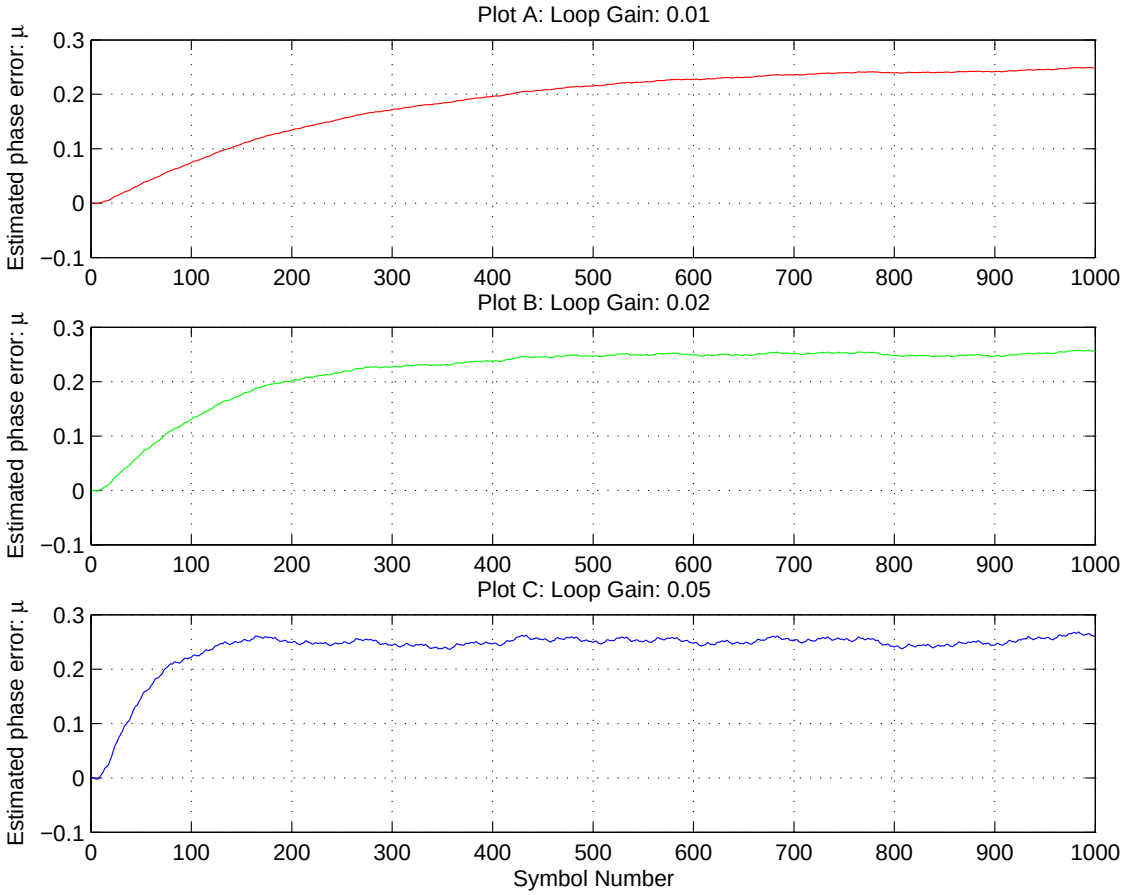

Figure 4.20: DPLL estimation of phase error illustration the rate of phase acquisition for a DPLL employing cubic interpolation for loop gains of 0.01, 0.02 and 0.05

In figure 4.20, plot A shows for an open loop gain of 0.01 at least 600 symbols are required to achieve acquisition. Plot B with an open loop gain of 0.02 roughly halves the number of required symbols required for acquisition. Plot C shows when the loop gain is increased to 0.05, the acquisition time is reduced by a factor of a half again taking approximately 150 symbols to return a phase error estimation in the order of $0.25 \times \pi$ radians.

In figure 4.21, the same received signal used in the simulation for the calculations of figure 4.20 is used for the simulation with the DPLL loop employing the linear interpolator.

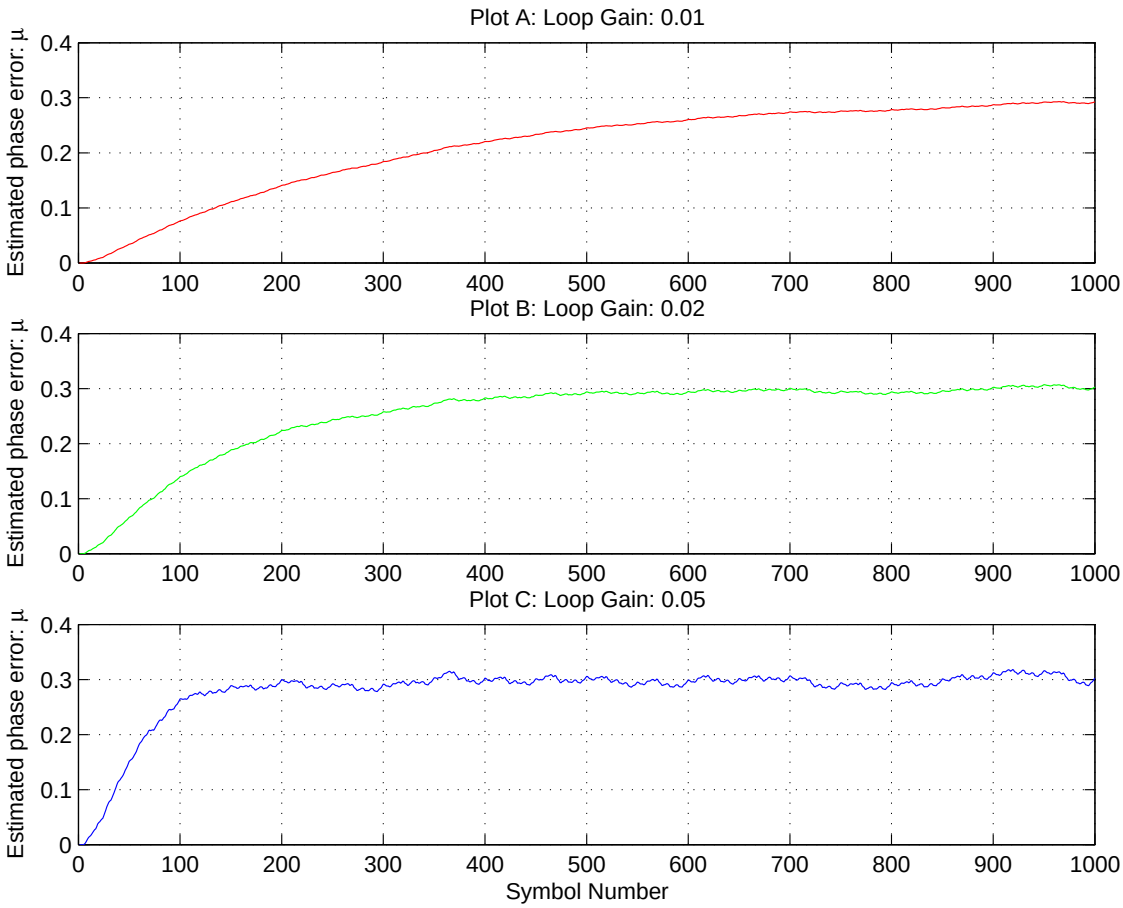

Figure 4.21: DPLL estimation of phase error illustration the rate of phase acquisition for a DPLL employing linear interpolation for loop gains of 0.01, 0.02 and 0.05

One of the outcomes of these two simulations is the linear interpolator produces an over-estimation of the phase error, as it is evident from figure 4.21 that the curves of the respective simulations approach $0.30 \times \pi$ as opposed to the actual phase of the received signal of $0.25 \times \pi$.

The plot of the linear interpolator indicates a performance similar to that of the cubic interpolator version, with the cubic interpolator acquiring the phase offset slightly more quickly.

4.3.3 Performance in White Gaussian Noise Channel

Assessment of the DPLL performance in a channel corrupted by AWGN is undertaken by first comparing the cubic and linear interpolator versions with the same simulated received signal data of 10db SNR, 15db SNR and 20dB SNR. The cubic version is then

investigated further using varying loop gain values related against the Bit Error Rate of the DPLL symbol decision block. The phase error of the simulated received signal to be acquired and tracked in this section is 0.25T.

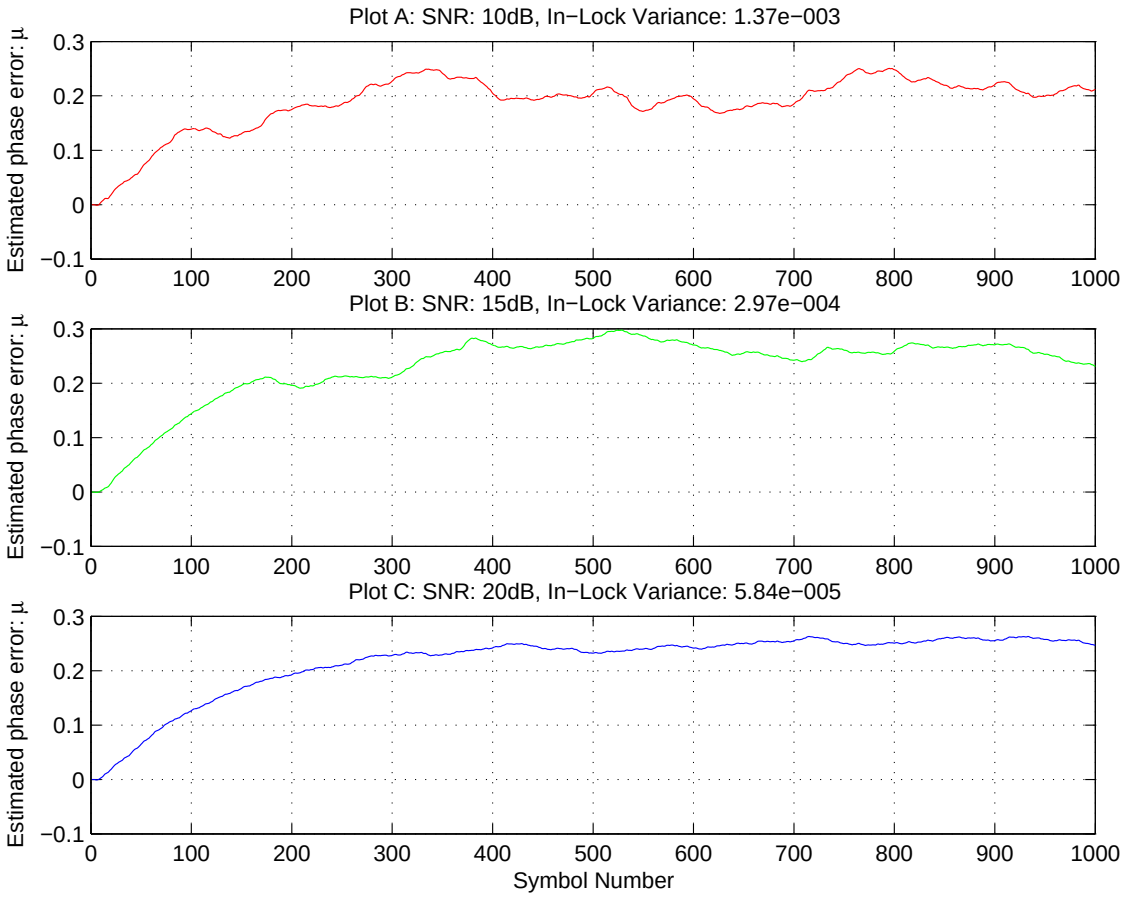

Figure 4.22: DPLL estimation of phase error illustration the rate of phase acquisition for a DPLL employing cubic interpolation for a loop gain of 0.02 for varying SNR levels aquiring and tracking a phase of 0.25π.

Plot A of figure 4.22 shows the phase being acquired and track with a variance of 1.37×10^{-3}. As the SNR is increased by 5dB in plot B, the variance reduces by an order of magnitude as the phase is being tracked with a greater degree of accuracy. This trend continues in plot C where the variance is again reduced in the by an order of magnitude as the SNR is again increased by 5dB.

Figure 4.23 shows the performance of the DPLL using a linear interpolator, the same simulated received data as was used in figure 4.22 with the cubic interpolator.

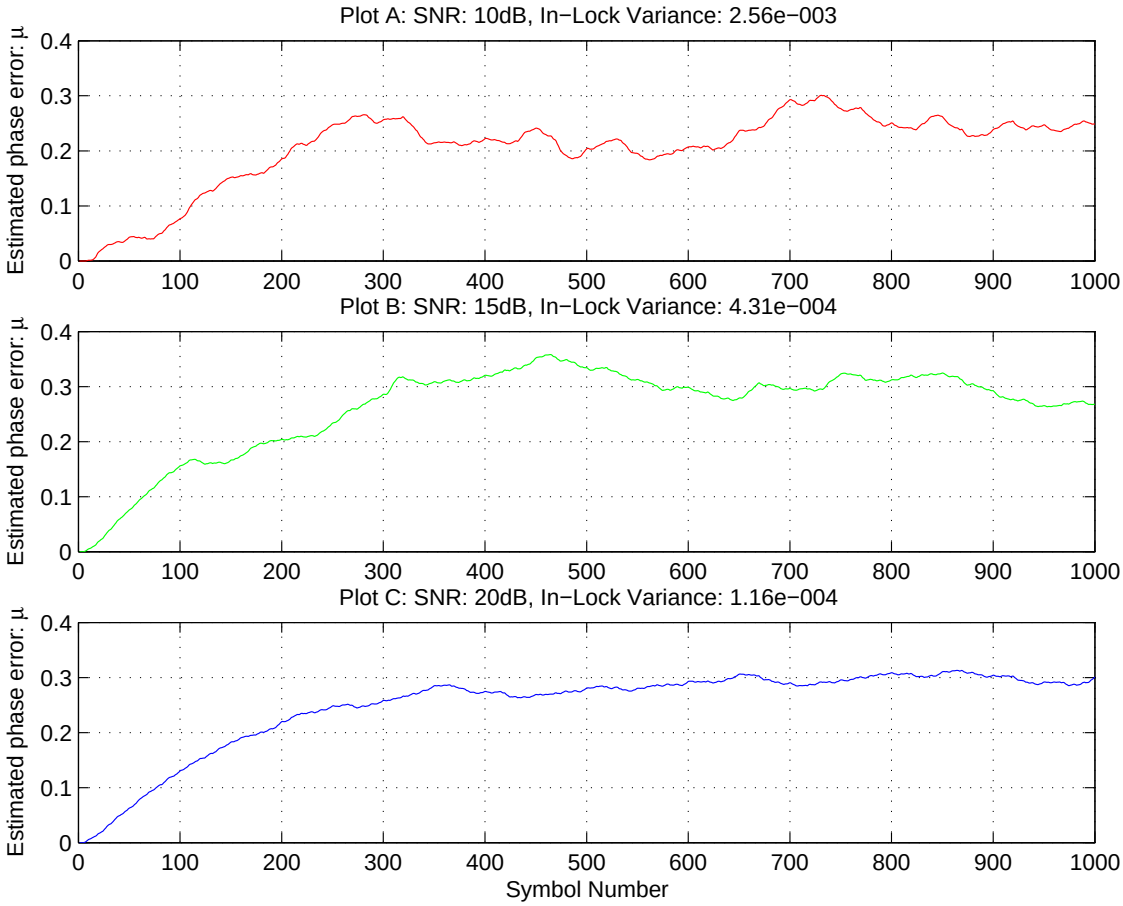

Figure 4.23: DPLL estimation of phase error illustration the rate of phase acquisition for a DPLL employing linear interpolation for a loop gain of 0.02 for varying SNR levels acquiring and tracking a phase of 0.25π.

When the linear interpolation version of the DPLL illustrated in the phase acquisition and tracking curves of figure 4.23 shows the variance is roughly doubled for each of the corresponding curves.

Figure 4.24 shows the phase acquisition and tracking behavior for a DPLL using cubic interpolation in an AWGN channel at 10dB SNR and with a loop gain of 0.01.

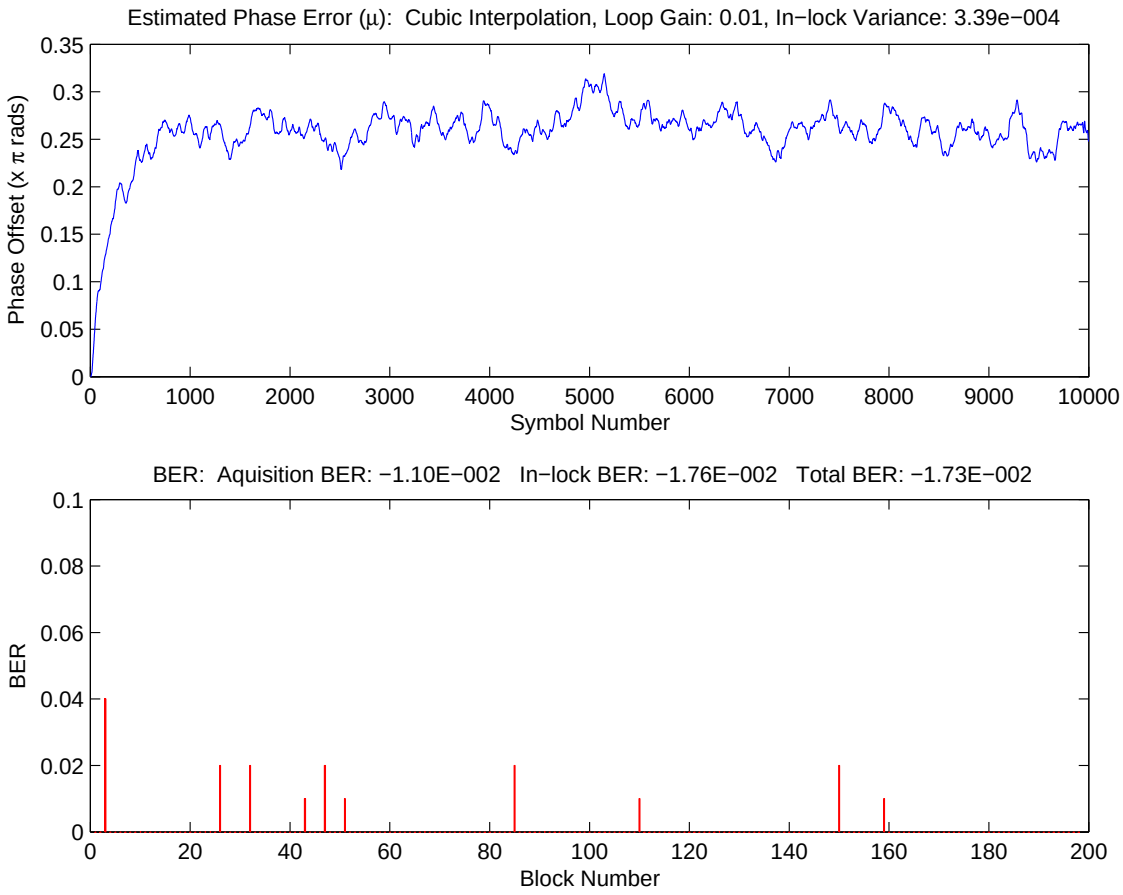

Figure 4.24: Cubic interpolator operating in an AWGN channel of 10dB SNR and a loop gain of 0.01. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

The BER plot which is illustrated in the lower pane of 4.24 shows a small error due to acquisition, and then several error spikes when the DPLL is in phase lock. The overall average BER for 200 blocks is calculated at 2.3×10^{-2} for a loop variance of 4.77e-4.

The loop gains is increased to 0.02 for the same simulated received data and the DPLL performance illustrated in figure 4.25.

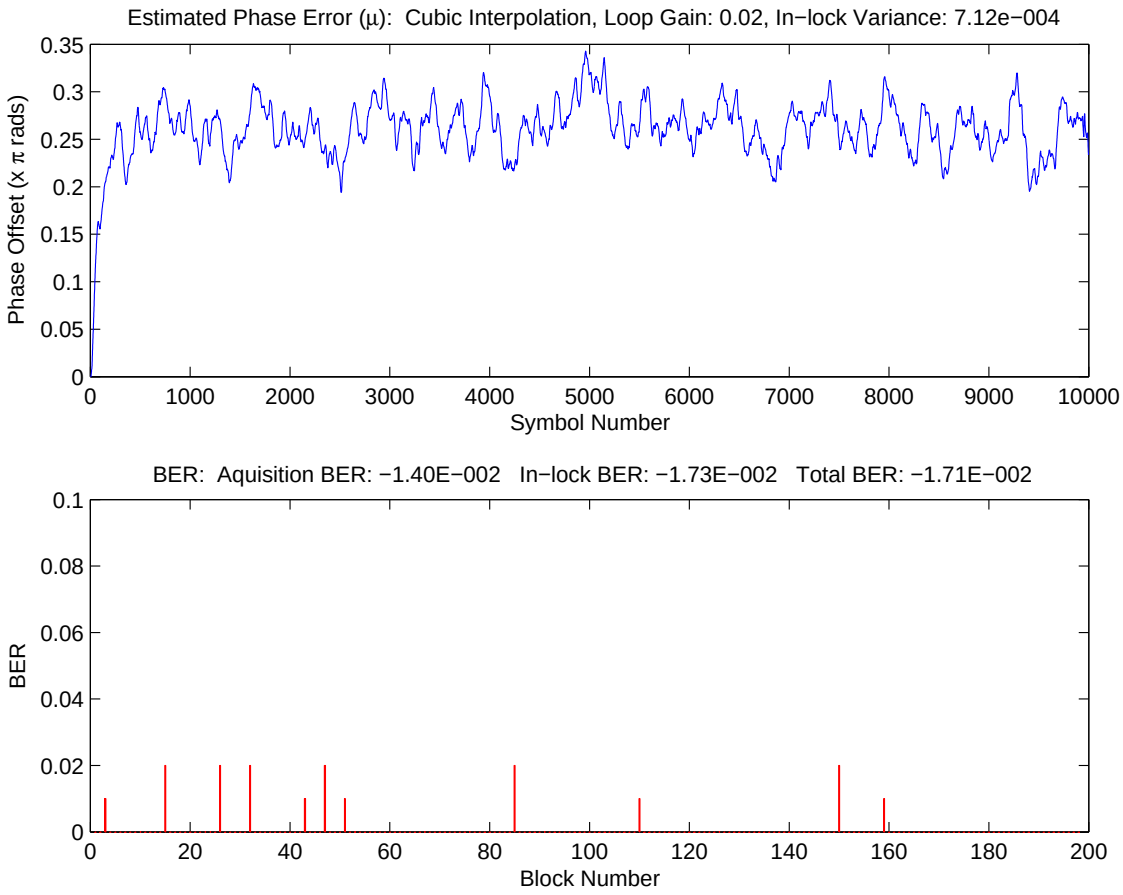

Figure 4.25: Cubic interpolator operating in an AWGN channel of 10dB SNR and a loop gain of 0.02. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

The BER profile of figure 4.25 is similar to that of figure 4.24, with the average BER per block reduced slightly to 1.8×10^{-2}, in spite of the the variance increasing by roughly a factor of 2.

In figure 4.26 the loop gain is increased to 0.05.

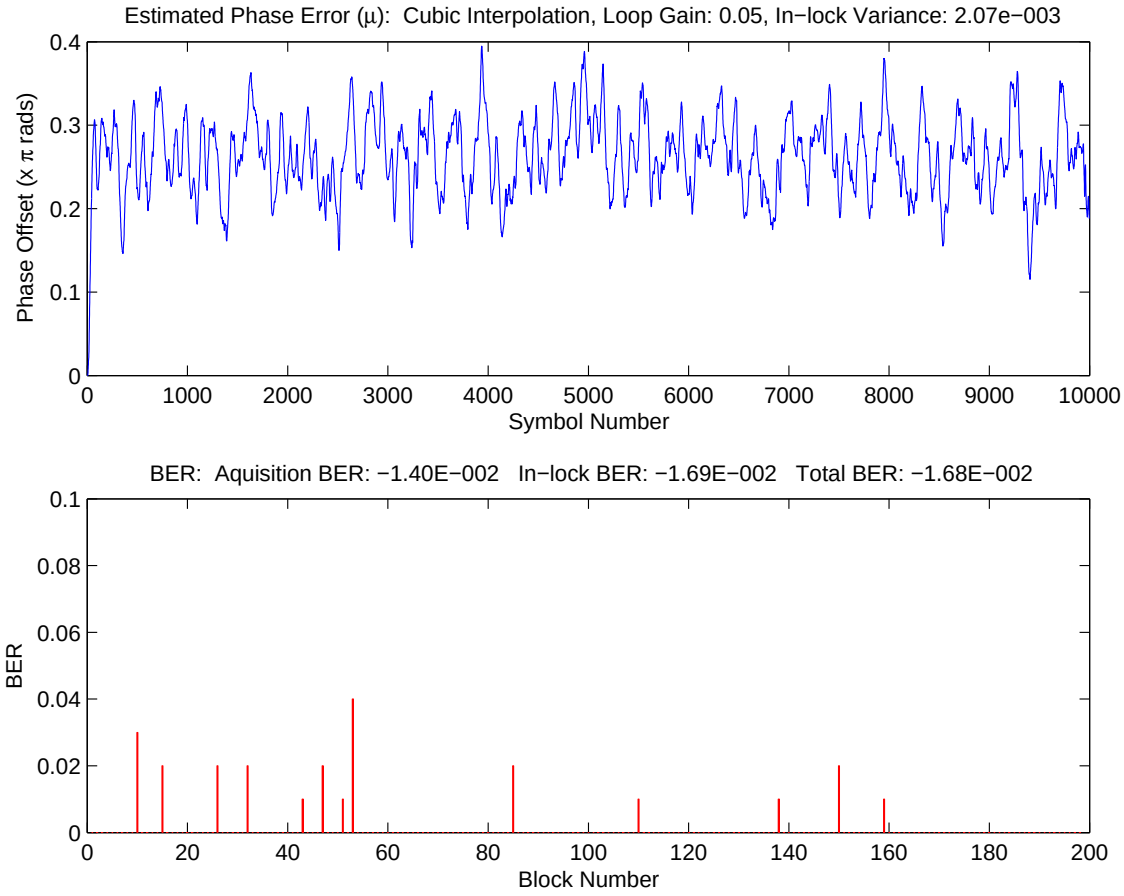

Figure 4.26: Cubic interpolator operating in an AWGN channel of 10dB SNR and a loop gain of 0.05. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

In figure 4.26 even though the variance has increased with the respective increase in loop gain, the BER profile has remained similar to that of figure 4.25, an average BER per block of 1.9×10^{-2}. The decision making ability of the DPLL is able to withstand what seems like a high degree of variance in phase estimation.

4.3.4 Performance in Flat Fading Channel

This section presents the performance of the DPLL when the channel is corrupted by flat fading. In this section only the cubic interpolator version of the DPLL is considered. The phase error of the simulated received signal to be acquired and tracked in this section is 0.25.

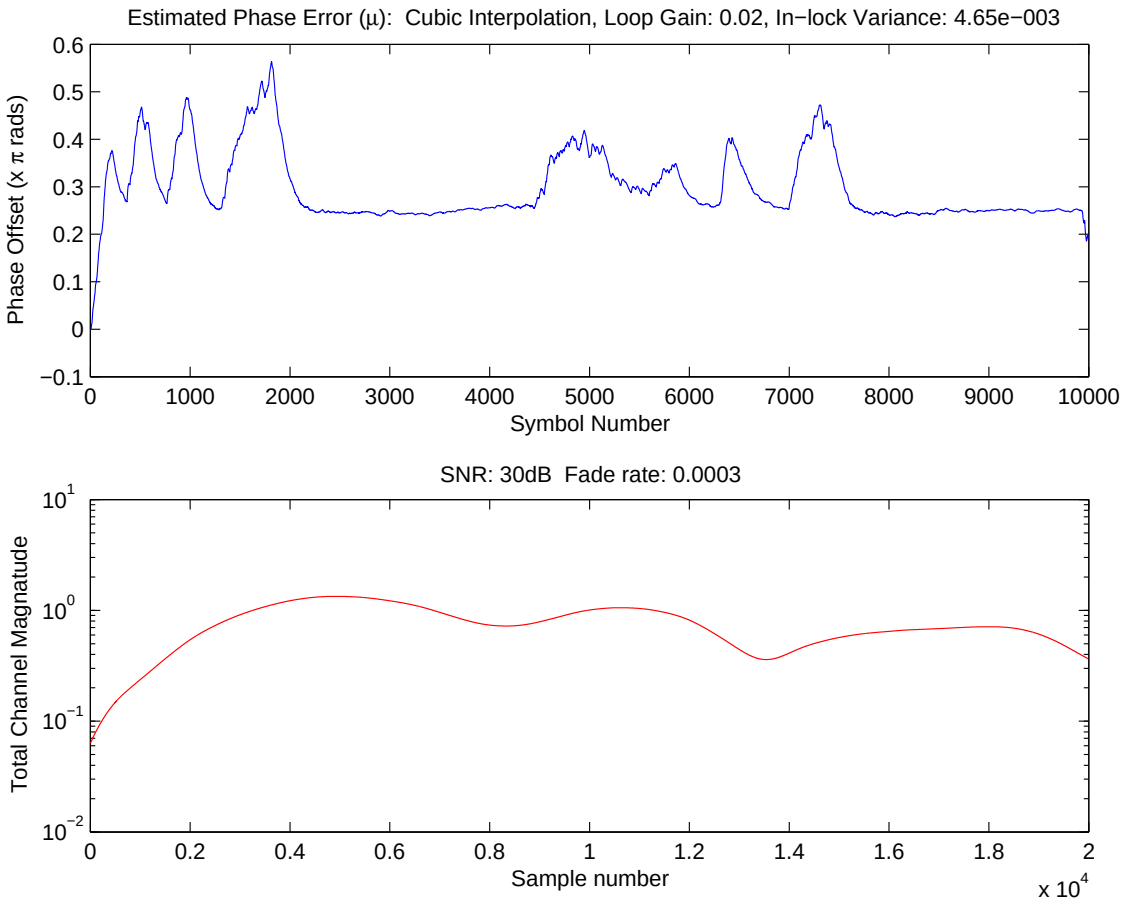

Figure 4.27: Cubic interpolator operating in an flat fading channel of f_{dT} of 0.0003 and 30dB SNR and a loop gain of 0.02. The total channel magnitude is illustrated in the lower plot.

Figure 4.27 illustrates the phase tracking profile for a relatively slow normalised fade rate of 0.0003. The phase of 0.25 appears to be acquired and tracked, with the profile showing a number of spikes that results in a variance of the profile of 4.65×10^{-3}.

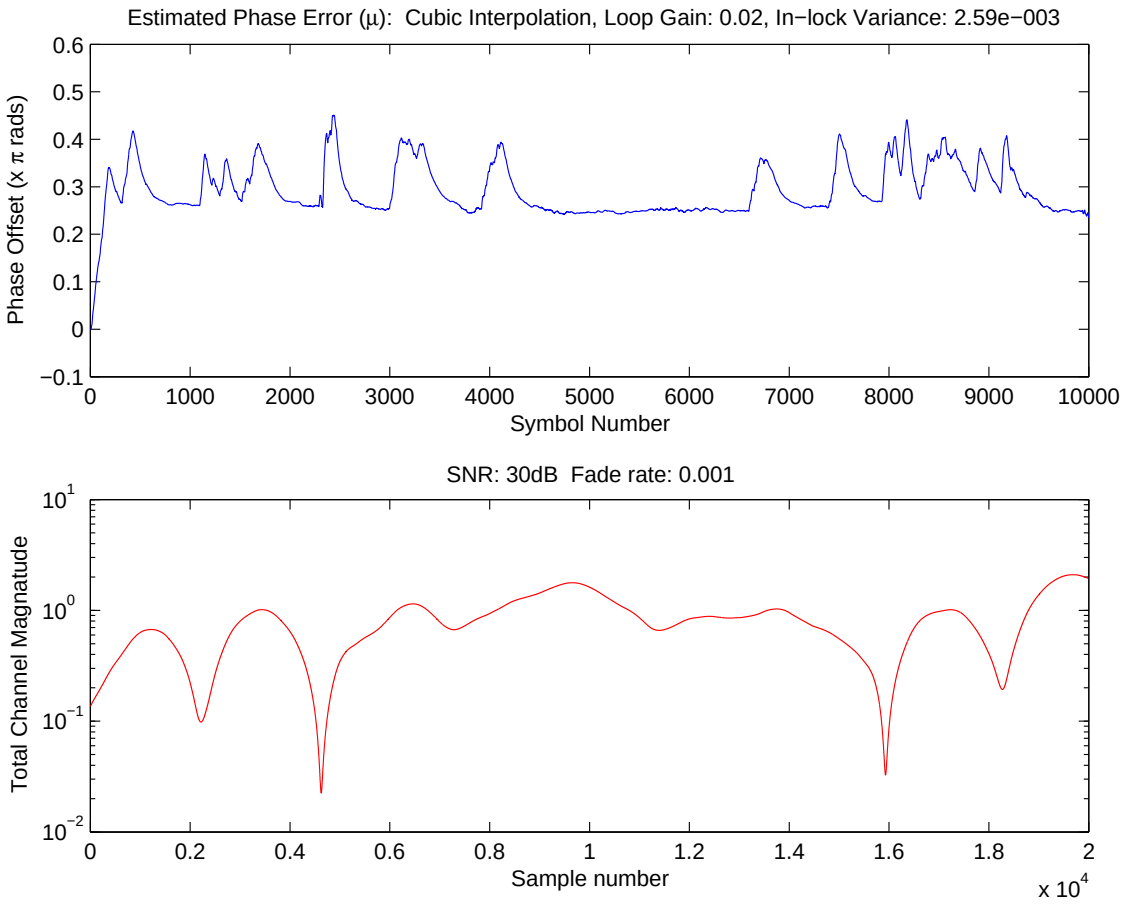

Figure 4.28: Cubic interpolator operating in an flat fading channel of f_{dT} of 0.001 and 30dB SNR and a loop gain of 0.02. The total channel magnitude is illustrated in the lower plot.

In figure 4.28 the fade rate is increased from 0.0003 to 0.001. As with figure 4.27 the phase is acquired and tracked, but with similar spikes occurring. For the faster fade rate, the variance is reduced to 2.59×10^{-3}.

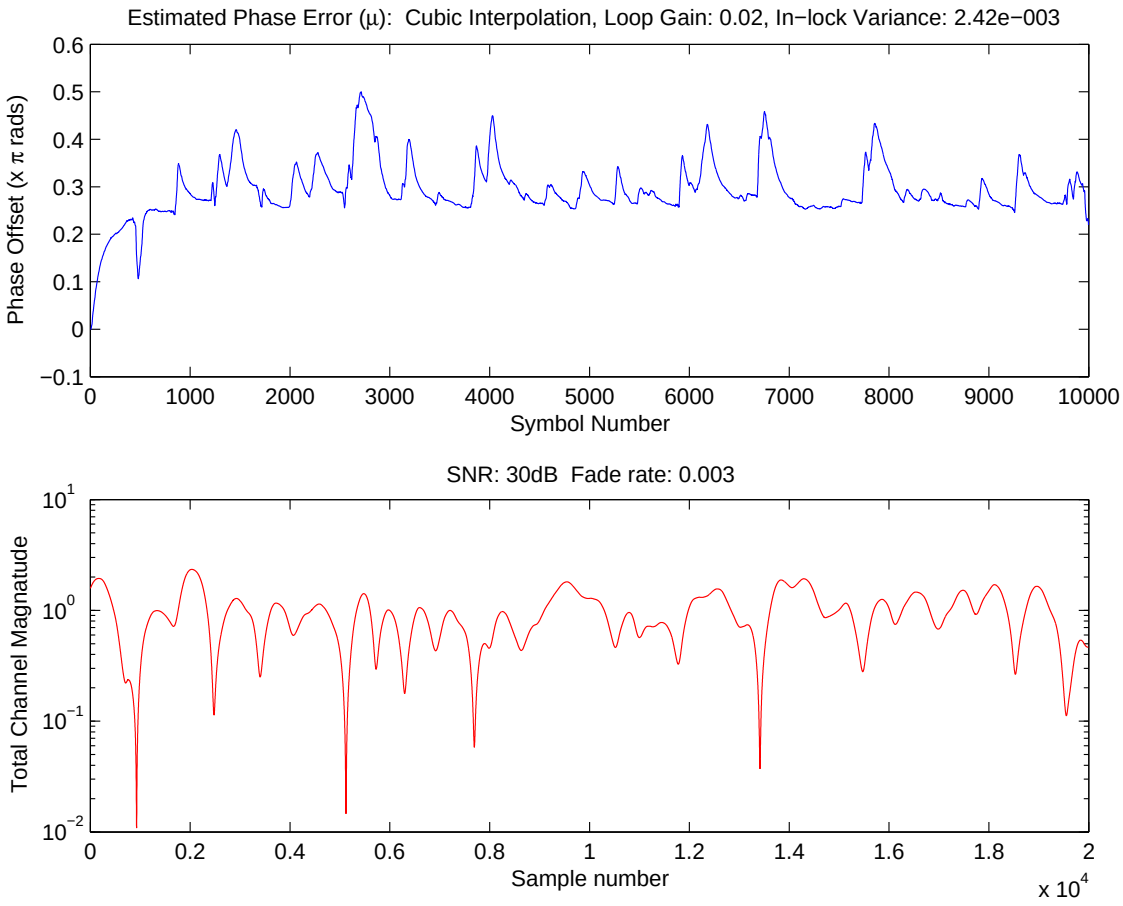

Figure 4.29: Cubic interpolator operating in an flat fading channel of f_{dT} of 0.003 and 30dB SNR and a loop gain of 0.02. The total channel magnitude is illustrated in the lower plot.

Figure 4.29 illustrates the estimated phase profile when the fade rate is increased to a relatively high covariance value of 0.003. Again the profile follows a curve similar to that of the two other figures in this subsection. The trend of reduced variance of the estimated phase error continues as the variance in this case is 2.42×10^{-3}.

4.4 Summary

The software design of the DPLL implementation for versions employing both linear and cubic interpolation was presented. The design was presented in functional block format, representing the implementation methodology. Each of these blocks was coded into Motorola ASM56000 assembly language for maximum execution speed. In all sections of this chapter the DPLL implementation under investigation was the decision directed

variant.

The software performance was measured using simulated received signals created by MatLab script. The simulated received signal was created with predetermined levels of additive white Gaussian noise (AWGN) and/or flat fading. The same signal could be used to test varying levels of open loop gain for the DPLL, along with linear or cubic modes of interpolation.

It was found that the cubic interpolator version would acquire phase slightly faster than the linear interpolator version in low noise conditions. In sections 4.3.2 and 4.3.3, the linear version showed a slight inaccuracy in phase calculation when in phase-lock mode, returning an acquired phase of about 0.3π radians for a simulated signal generate with an error of 0.25π radians.

When AWGN is introduced, similar phase acquisition results were produced. When the variance of the DPLL estimated phase error is measured, the linear interpolator version has approximately double the variance for that of the cubic version receiving the same signal.

The error rate of the symbol decision block of the DPLL was measured against the original symbols, using differential encoding and decoding the BER per 100 bit block was produced. For the DPLL using cubic interpolation in an AWGN channel, loop gain was increased from 0.01 to 0.05, producing a estimated phase error variance from 4.77×10^{-4} to 2.45×10^{-3}. Over this range the BER remained fairly stable at a rate of around 2.0×10^{-2}, indicating the DPLL's ability to withstand what seems like a high level of variance in phase tracking.

DPLL performance was then measured in a flat fading channel. At low normalised fade rate, the phase was acquired and tracked with similar performance of the DPLL operating in a channel with a low level of SNR (10dB), but suffered some tracking *spikes*. As the fading variance is increased, the incidence of these spikes is increased but the magnitude is reduced.

Chapter 5

A Real-Time Coherent Receiver Using D-QPSK

This chapter describes the hardware configuration, the DSP software development and some real-time operational DPLL performance results achieved as a part of this thesis.

Figure 5.1: Photo of Experimental Setup

Figure 5.1 shows an overall photograph of the experimental platform used throughout the hardware and software development phase of the project. The first section of this chapter provides an overall system view, followed by a brief description of each sub-system;

the transmitter, channel and receiver. The second section describes each of the electronic hardware components, with detail of performance and capability. Sections three, four and five then describe the software implementation and hardware configuration of the transmitter, communication channel and receiver front end. Section seven presents operational performance results of the DPLL for various levels and types of channel corruption.

5.1 Experimental Platform Overview

The test system incorporated two Motorola 24-bit DSP56321 EVM boards, a transmitter board and a receiver board. These hardware blocks are described in further detail in section 5.2.

The configuration of the hardware blocks, along with applicable clock sources is depicted in figure 5.2.

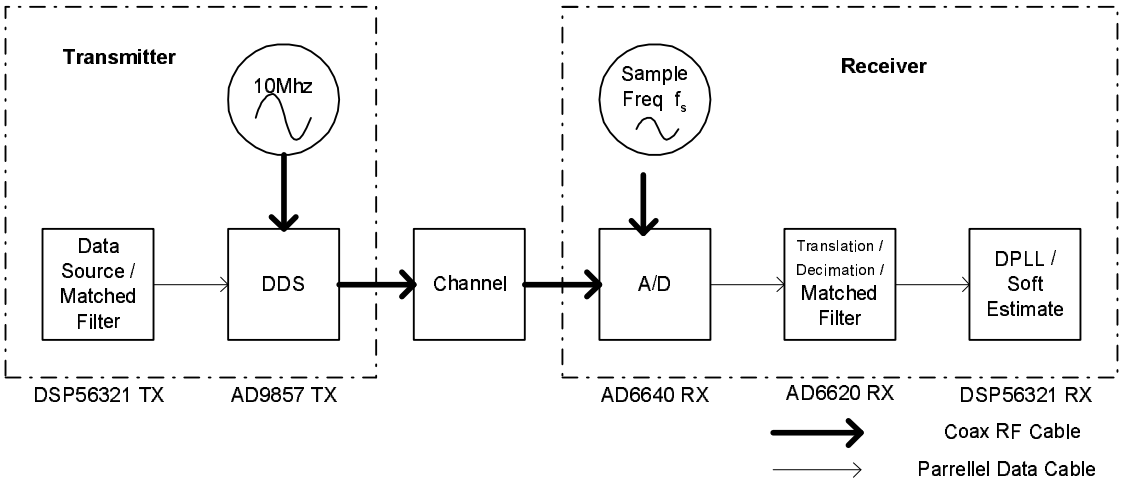

Figure 5.2: Experimental configuration for Digital Transmission

The DSP56321 EVM boards were programmed with DSP code written specifically for the purposes of this thesis along with the AD9857 based waveform synthesizer and AD6620 based A/D, decimation and matched filter boards being reconfigured as required.

For the purposes of this thesis the AD9857 board in the transmitter will be termed the AD9857 TX board, the DSP56321 board in the transmitter the DSP56321 TX board, the AD6620 board in the receiver will be termed the AD6620 RX board and the DSP56321 board in the receiver the DSP56321 RX board. The receiver also includes an AD9857

based programmable clock source that provides the AD6620 RX board with clock at the required sampling rate f_s.

5.1.1 Transmitter Overview

As shown in Figure 5.2, the transmitter was constructed using a Motorola 56321 DSP board, an AD9857 based DDS board and a stable 10 MHz clock source.

The 56321 TX board produces an over sampled, complex baseband digital signal *pulse shaped* by a Square Root Raised Cosine (SRRC) pulse-shaping filter with a roll-off of 0.35. These complex baseband values are communicated to the AD9857 TX board by means of an external parallel data bus. The AD9857 TX board then performs Direct Digital Synthesis (DDS) to modulate 125k D-QPSK Symbols per second at a 45MHz carrier frequency.

5.1.2 Channel Overview

Two sets of equipment are alternated to simulate either an AWGN channel or a fading channel as required.

The AWGN channel is simulated by using a variable attenuation block to degrade the signal power available at the receiver and hence the available SNR. Simulating the fading channel is a more complex process, achieved through the use of an HP 11759B fading channel simulator pre-programmed with varying Doppler fading frequencies, path length differences and path attenuations.

5.1.3 Receiver Overview

The receiver consists of an AD6640 analog to digital converter (A/D) and an AD6620 digital down-converter that communicates with the DSP56321 board via an external parallel data bus.

The AD6620 RX board performs bandpass sampling of the 45MHz QPSK RF signal at a sampling rate (f_s) of 40MHz. The AD6620 RX Board also performs the SRRC matched filtering and sampling rate decimation and ultimately presents the DSP56321 RX board with complex baseband samples at a rate of 2 samples/symbol. The DSP56321 RX board

then iteratively estimates the phase of the incoming signal, interpolates the signal at the optimal timing point in order to produce a soft output (symbol estimate) at the symbol rate. The soft output and phase error in the test configuration (as used in this thesis) are saved to DSP56321 RX on-board memory for later download and post-processing by an external computer.

5.2 Hardware Description

A description is given in this section of the various hardware items utilized in the construction of the experimental platform. Three Analog Devices based boards are used for direct digital synthesis, A/D, decimation and matched filtering the AD9857, AD6640 and AD6620 respectively. The DSP implemented functionality includes; software based implementation of a symbol source, pulse shaping filter and digital phase locked loop (DPLL).

5.2.1 Motorola DSP56321 Digital Signal Processor EVM Board

The functional block diagram of the 56321 Digital Signal Processor (DSP) is reproduced [34] in Figure 5.3.

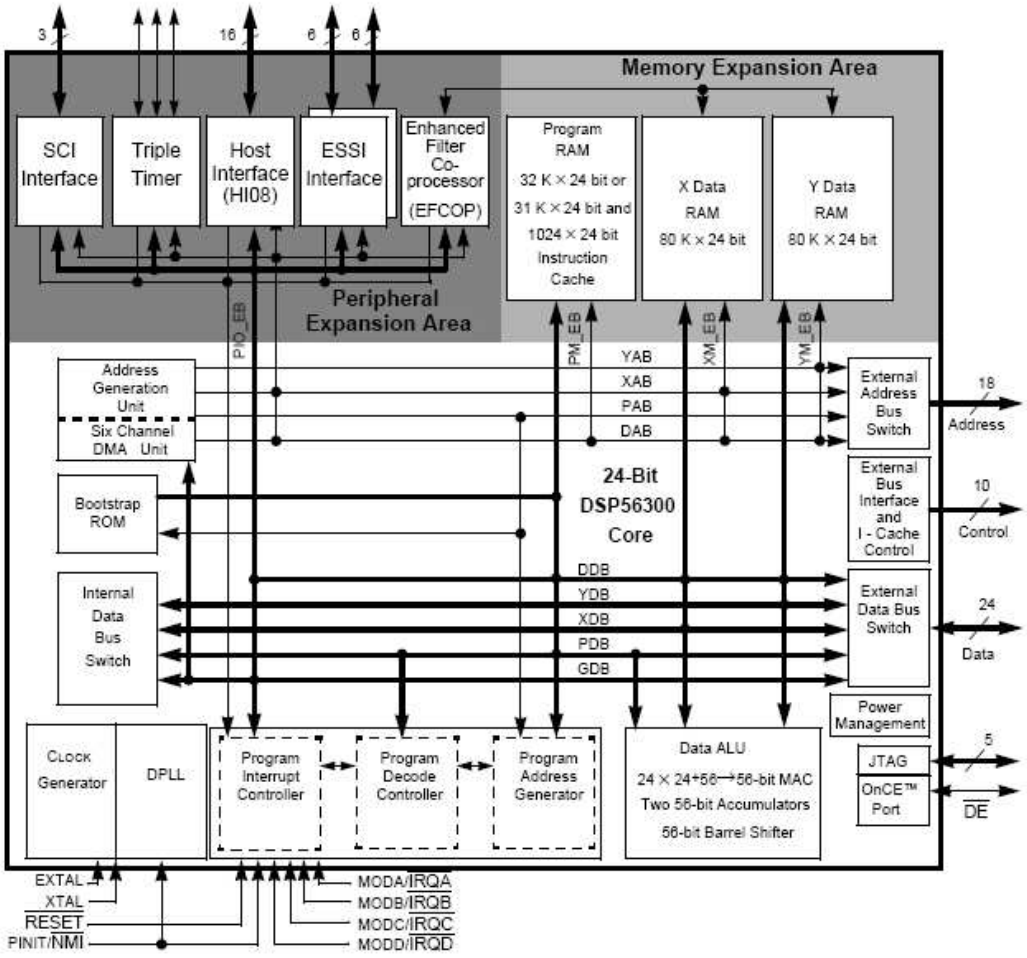

Figure 5.3: DSP56321 Schematic [34]

The Motorola DSP56321 is a multi purpose DSP capable of 275 million multiply-accumulates per second (MMACS). It has a Data arithmetic logic unit (Data ALU) with fully pipelined 24 x 24-bit parallel multiplier-accumulator (MAC) and 56-bit parallel barrel shifter for high speed normalization among other built in functionalities.

A very useful feature of the DSP56321 is the Enhanced Filter Coprocessor (EFCOP); a fully programmable Internal 24 x 24-bit filtering and echo-cancellation coprocessor that runs in parallel to the DSP core effectively raising the overall DSP processing rate to 550 MMACS.

A total of 192k x 24bit internal memory addresses are available, configurable into program memory, x memory and y memory blocks.

5.2.2 Analog Devices AD9857 Direct Digital Synthesis Hardware

The functional block diagram of the AD9857 Quadrature Digital Upconverter is reproduced [12] in Figure 5.4.

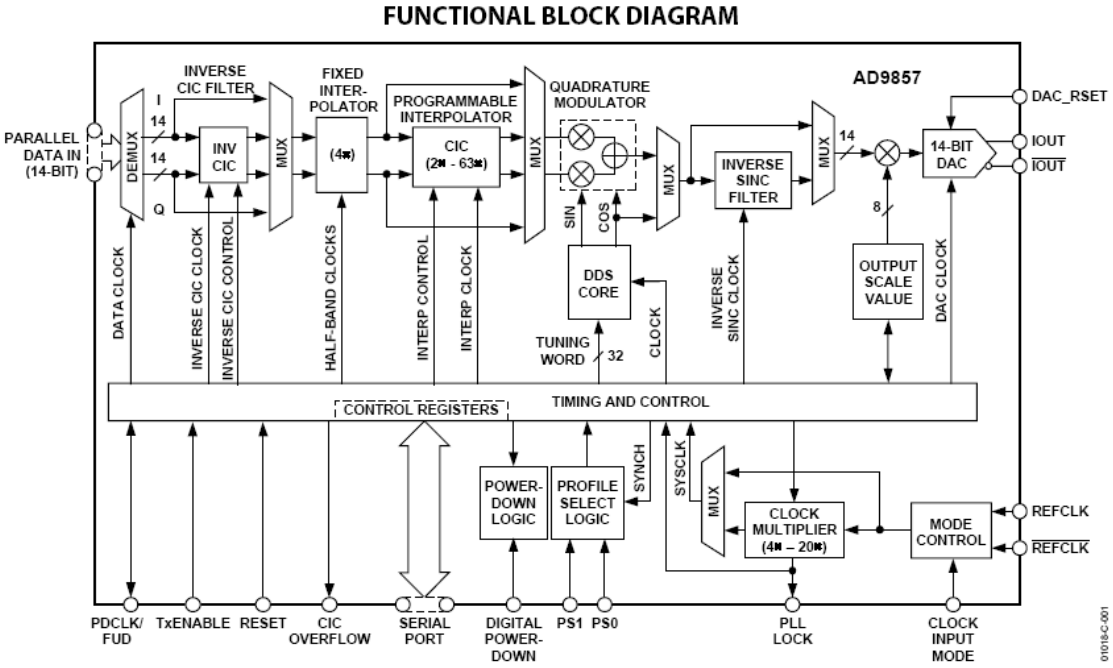

Figure 5.4: AD9857 Quadrature Digital Upconverter [12]

The Analog Devices AD9857 chipset incorporates a high-speed direct-digital synthesizer (DDS), a high-speed 14-bit digital-to-analog converter (DAC), a configurable single tone clock source to form a complete quadrature digital upconverter device capable as functioning as a universal I/Q modulator.

The AD9857 operates with a 200MHz internal clock source, a 14-bit data path and correction filtering.

5.2.3 Analog Devices AD6640 Analog to Digital Conversion Hardware

The functional block diagram of the AD6640 Digital to Analog converter is reproduced [35] in Figure 5.5.

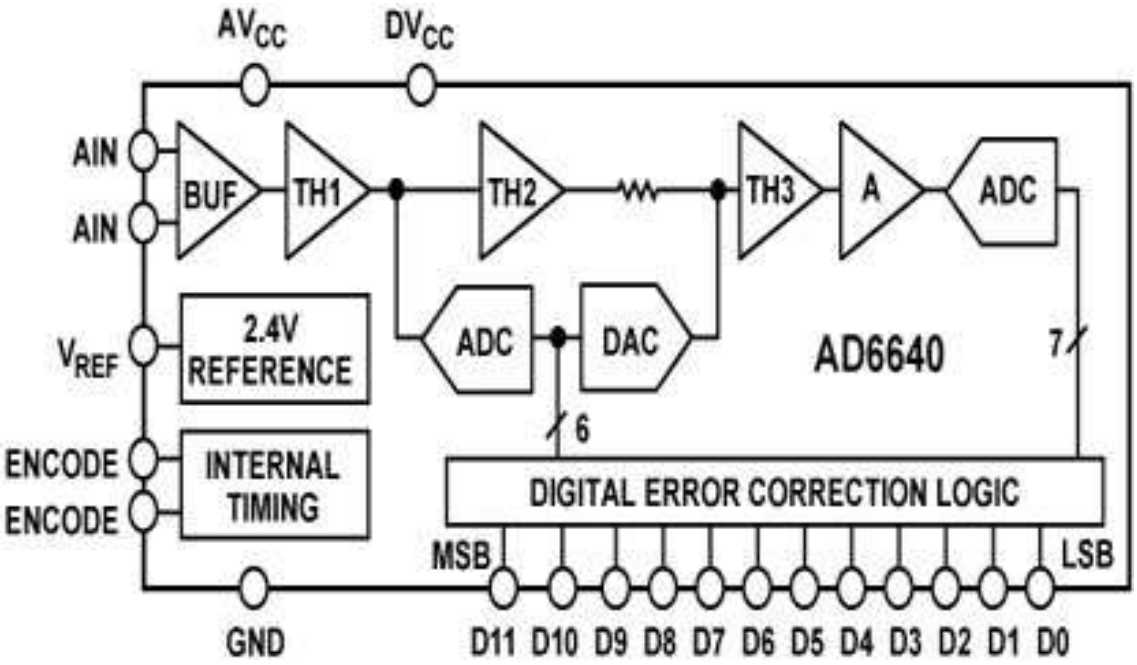

Figure 5.5: AD6640 Analog to Digital Converter [35]

The AD6640 is a wideband analog-to-digital converter capable of sampling at 65 MSPS. High dynamic range wideband sampling enables the AD6640 to digitize up to 25 MHz of spectrum at one time.

5.2.4 Analog Devices AD6620 Frequency Translation, Decimation and Matched-Filter Hardware

The functional block diagram of the AD6620 Digital Receive Signal Processor is reproduced [36] in Figure 5.6.

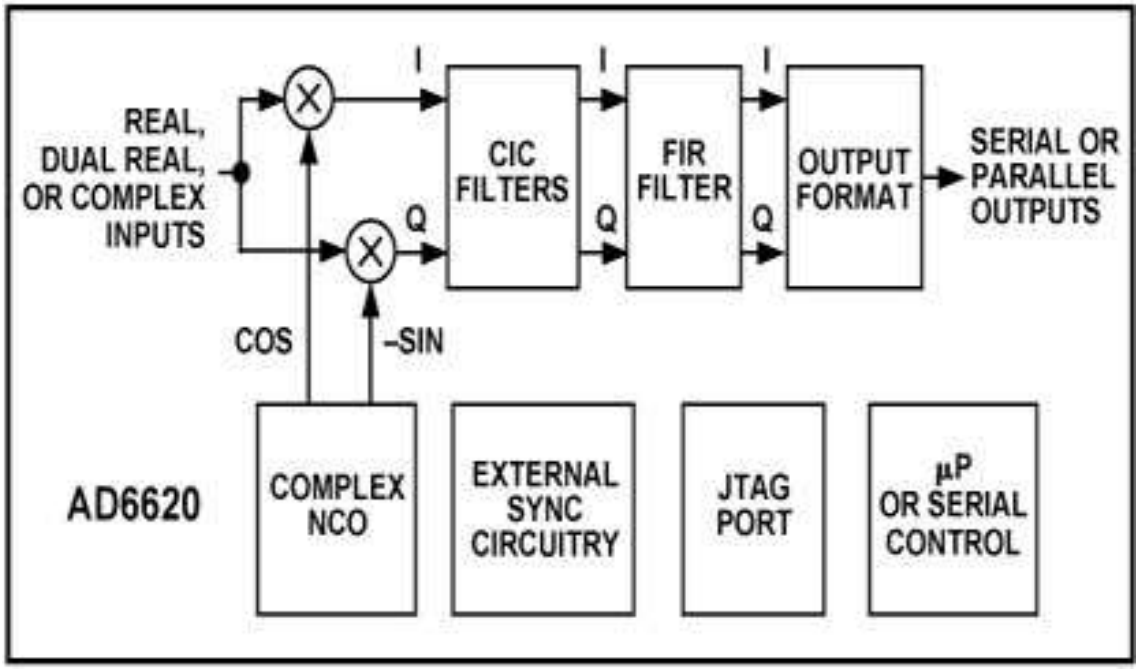

Figure 5.6: AD6620 Digital Receive Signal Processor [36]

The AD6620 can process up to 67 Million Samples Per Second (MSPS) in real channel mode, or 33.5MSPS in either diversity[1] or complex mode. It features an Numerically Controlled Oscillator (NCO) frequency translation function, a 2nd order Cascaded Integrator Comb FIR Filter with available decimation rates of 2, 3 . . . 16, a 5th Order Cascaded Integrator Comb FIR Filter Linear Phase, Fixed Coefficients with programmable Decimation Rates of 1, 2, 3 . . . 32.

The final stage of the AD6620 is a programmable, decimating FIR filter (that can be employed as a matched filter) capable of up to 134 Million Multiply-Accumulates (MMACs) per second of 256 20-bit coefficients and decimation rates of 1, 2, 3 . . . 32

5.3 Transmitter Implementation

Figure 5.7 shows the functional blocks of the DSP software and built in DSP functionality and how those blocks interact with hardware signals.

[1] Two independent real channels

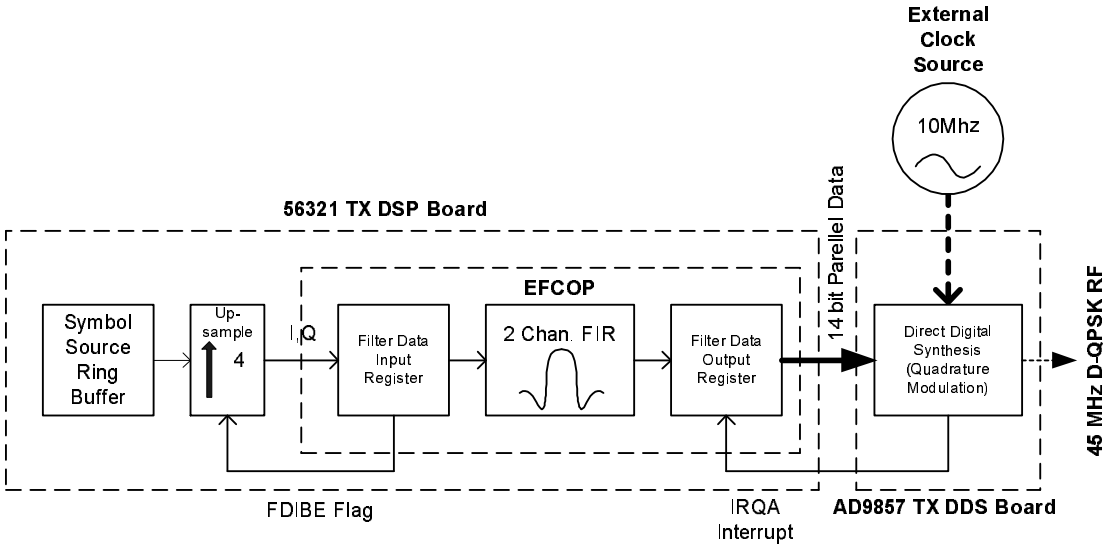

Figure 5.7: Transmitter Configuration

The symbol source provides data to the upsampling block, which then pushes data on request to the EFCOP filter for pulse shaping. The filter output register then pushes data onto the external data bus to the direct digital synthesis block.

The blocks of figure 5.7 are described in further detail in the following sub-sections.

5.3.1 Symbol Source

Initially a block of 100 bits of known random binary data is created and pre-processed to produce 50 D-QPSK encoded symbols. When the code for the DSP56321 TX board is compiled, this block of pre-encoded data is included in the DSP software as a ring buffer such that this block of 50 symbols is then repeatedly transmitted in a continual cycle.

5.3.2 Pulse Shaping

The Enhanced Filter-Coprocessor (EFCOP) used as a configurable on-chip filter lends itself very well to use as a pulse-shaping filter in this application.

In the configuration of Figure 5.7 the Filter Data Buffer Input Empty (FDIBE) flag of the EFCOP is continually polled, and when the flag is set then a new value is passed onto the Filter Data Input Register (FDIR). Values from the I,Q ring buffer are padded out

with zeros such that the signal is four times oversampled[2]. These oversampled I and Q values are then sequentially placed on the EFCOP FDIR for pulse shaping. Pulse shaping is achieved by pre-configuring the EFCOP as an interleaving 2-channel FIR filter with 64 shared coefficients of a SRRC pulse shape with a rolloff coefficient of 0.35.

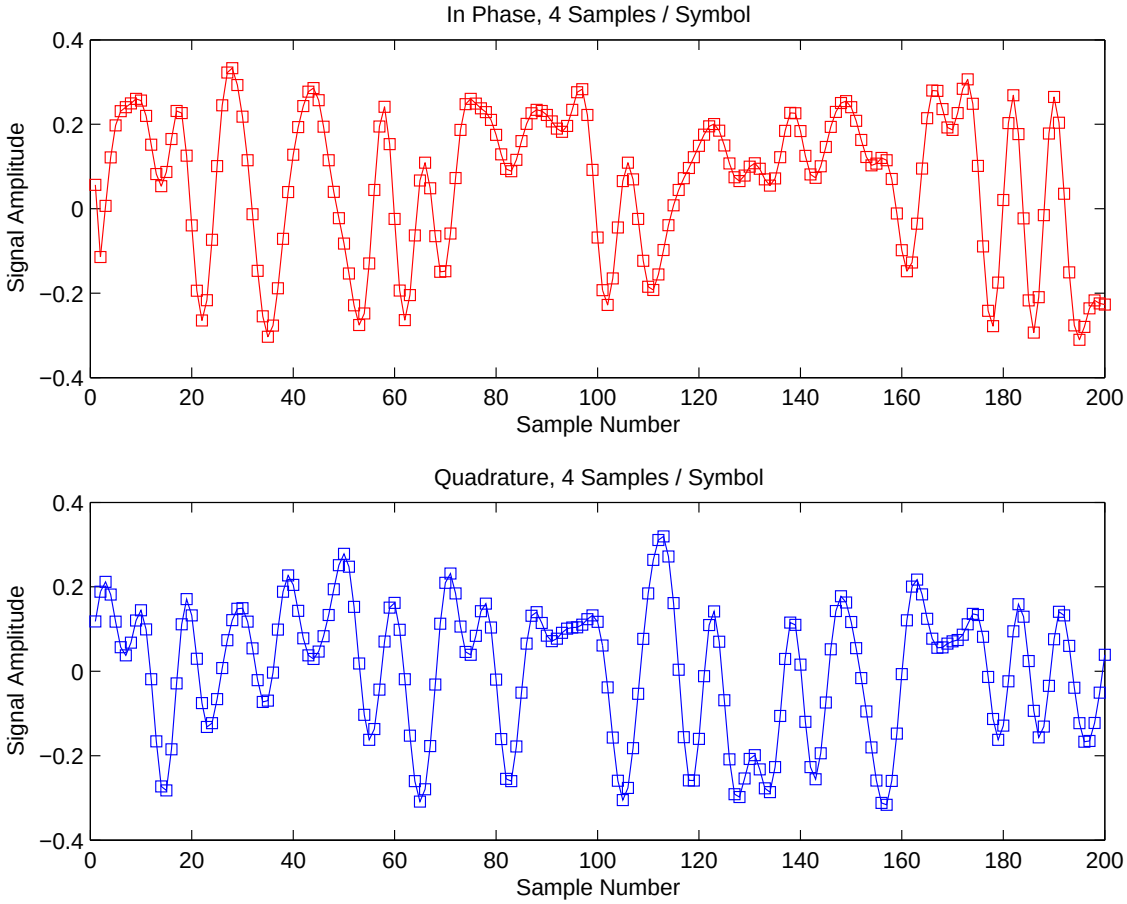

Figure 5.8: Baseband I & Q Waveforms

Figure 5.8 illustrates a short period of the baseband waveforms of real data as generated by the pulse shaping process of the EFCOP transmitter described above. It is these values that are passed by 14-bit external data bus to the AD9857 TX board.

5.3.3 Direct Digital Synthesis of the D-QPSK Waveform

The DSP56321 TX board acts to service data requests from the direct digital synthesis upconverter, an AD9857 TX. A request from the AD9857 TX is effected by invoking an

[2] As described in section 5.3.2

external interrupt (IRQA) on the DSP56321 TX, which then pulls the oldest value off the filter data output register (FDOR) and pushes it onto the external bus connected to the AD9857 TX for quadrature DDS modulation. The process of reading a byte off the FDOR then effectively clears a space on the FDIR, which prompts the next oversampled value of the symbol source ring buffer to be placed onto the FDIR.

The AD9857 TX is configured such that it produces a symbol rate of 125k Symbols / second at a carrier frequency of 45MHz. The waveform spectrum as observed by a spectrum analyser is illustrated below in Figure 5.9.

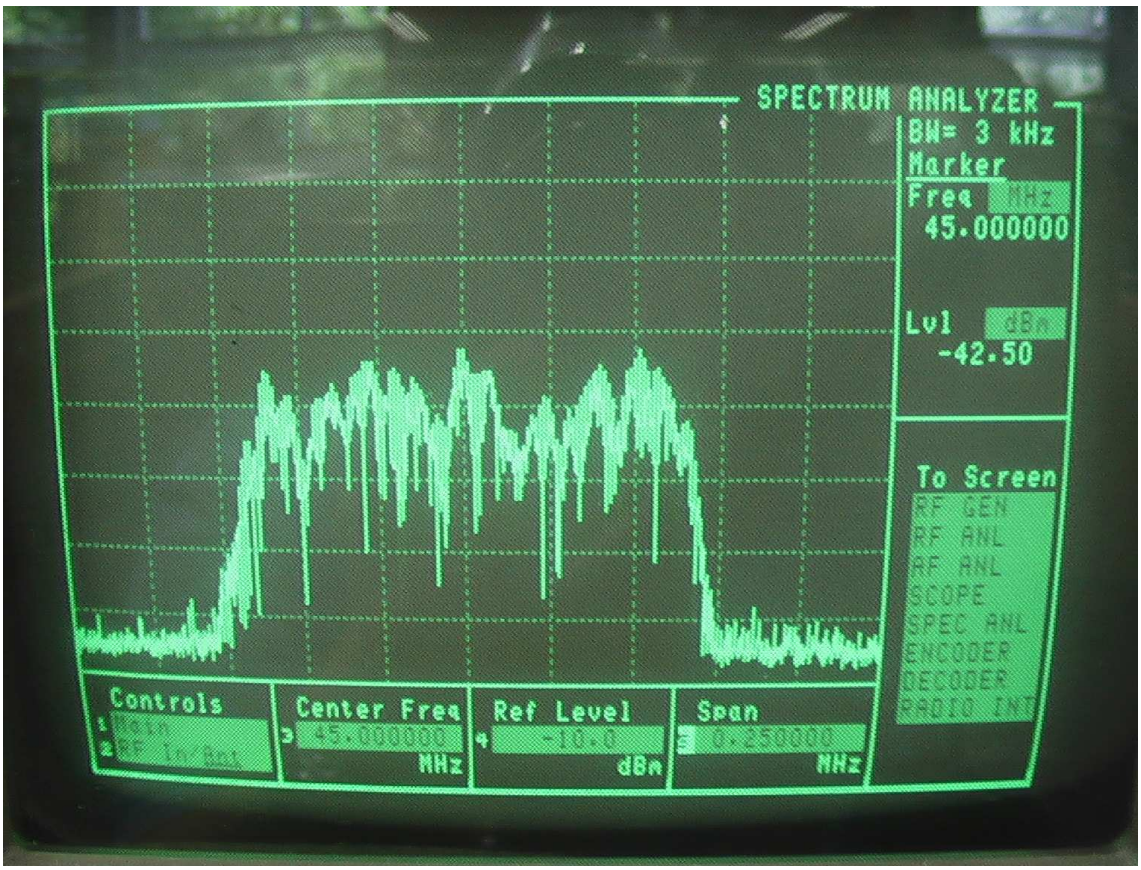

Figure 5.9: Transmitter Waveform Spectrum as Measured by HP Spectrum Analyser

The spectrum analyser display of Figure 5.9 shows the spectrum of a waveform with a bandwidth of 250kHz, produced by SRRC shaped pulses with a rolloff of 0.35 about a carrier of 45MHz.

5.4 Channel Simulation Implementation

Two types of channel corruption are simulated, the Additive White Gaussian Noise (AWGN) channel and the fading channel.

5.4.1 Additive White Gaussian Noise Channel

Simulation of an AWGN channel simulation case is a trivial case. The SNR ratio is set by attenuation of the carrier and reducing the ratio between the carrier power and the noise floor. The approximate SNR level of corruption experienced can be estimated by measuring the ratio between the carrier power and the noise floor directly off the spectrum analyser. A Gaussian noise source was not available in the frequency band of interest, which had precluded the opportunity for an ideal experimental implementation where the SNR is configured by the implementation of a controllable external AWGN noise source.

5.4.2 Fading Channel

A fading channel is simulated by using an HP 11759B Fading channel simulator.

The HP 11759B is configured with Doppler frequencies, attenuations and delay variables for three separate paths in order to produce a dispersive fading channel. An example set-up for a Fading channel with a Doppler frequency of 40Hz is illustrated in Figure 5.10.

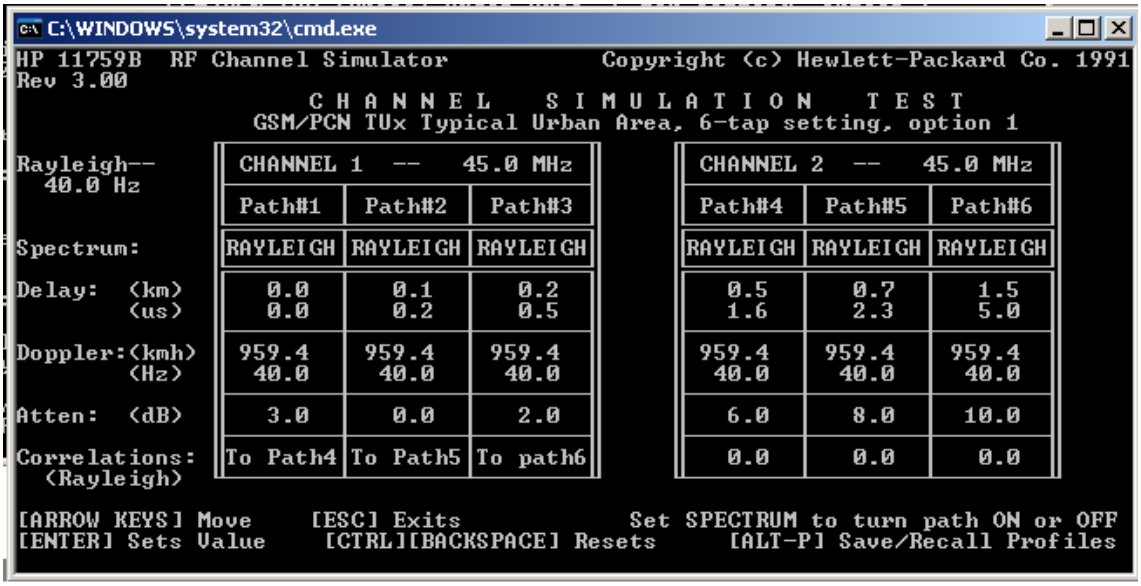

Figure 5.10: HP 11759B Simulator Configuration

A problem was experienced with the usage of the HP 11759B due to the input power requirement of 0dBm for the transmitted signal. An RF amplifier in the band of interest was not available and the optimal 0dBm level not achievable; a maximum transmission power configuration of -11dBm being the maximum level able to be produced. The effect from the lack of transmission power in this case manifests itself as a loss of dynamic range, such that when the signal enters a deep fade in the HP 11759B simulator the carrier signal may disappear below the noise floor rendering no signal recovery possible.

An illustration of channel characteristics created by the HP 11759B is presented below in order that the operation of the simulator may be visualised. A D-QPSK waveform at 45MHz was transmitted through the simulator, signal samples were collected from the receiver and then the received power calculated. The magnitude of each complex sample calculated and an averaging filter applied to smooth the profile.

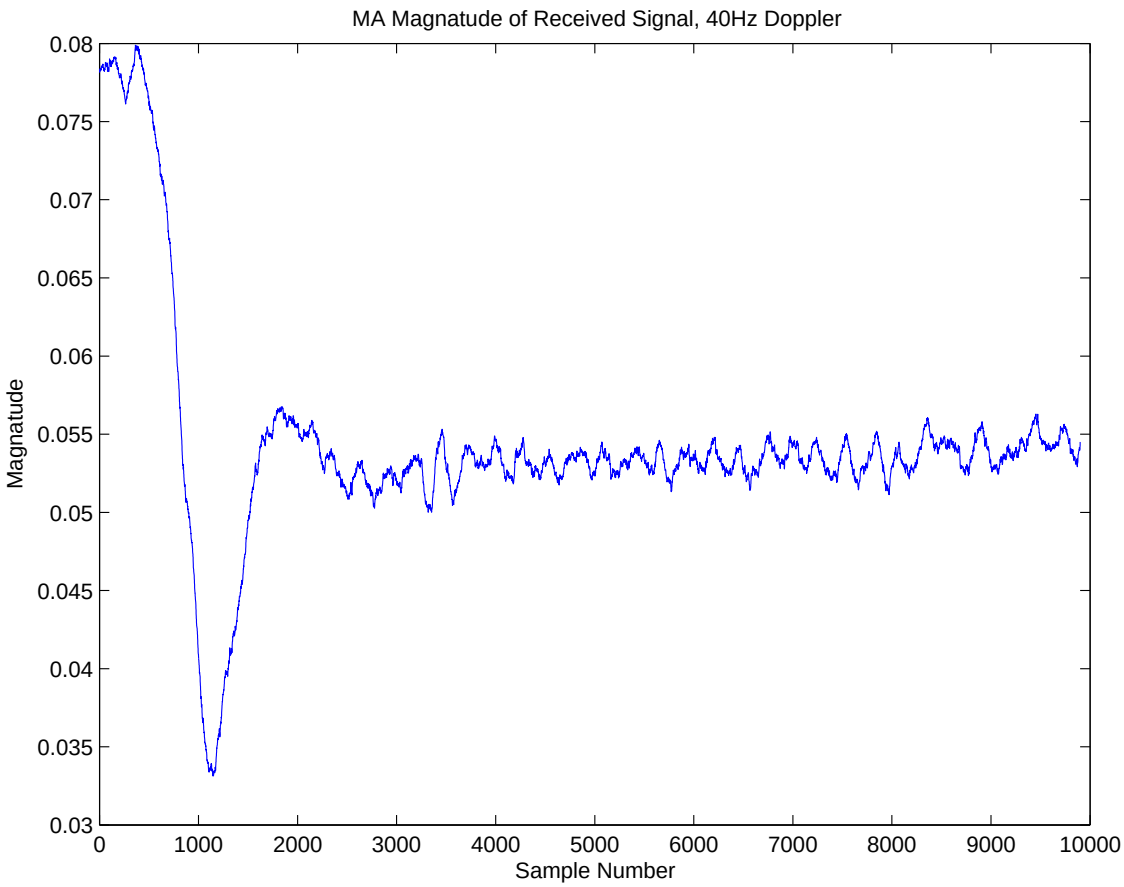

Figure 5.11: Fading Channel Profile: 40Hz Doppler Frequency

Figure 5.11 illustrates a fade occurring at around the 1,000th sample received, followed by smaller oscillations and a more stable channel for the remaining received samples. As the AGC function has already been applied, the overall attenuation of the channel is not apparent form these measurements. The phase distortion due to channel corruption from the HP 11759B is not shown.

5.5 Receiver Implementation

The receiver comprises A/D conversion, decimation, matched filtering and DPLL functions. The receiver front end was implemented using SASRATs receiver hardware [5], configured for this application as described in sections 5.5.1 and 5.5.2. The implementation of the DPLL is described in section 5.5.3 and the offline processing undertaken in section 5.5.4.

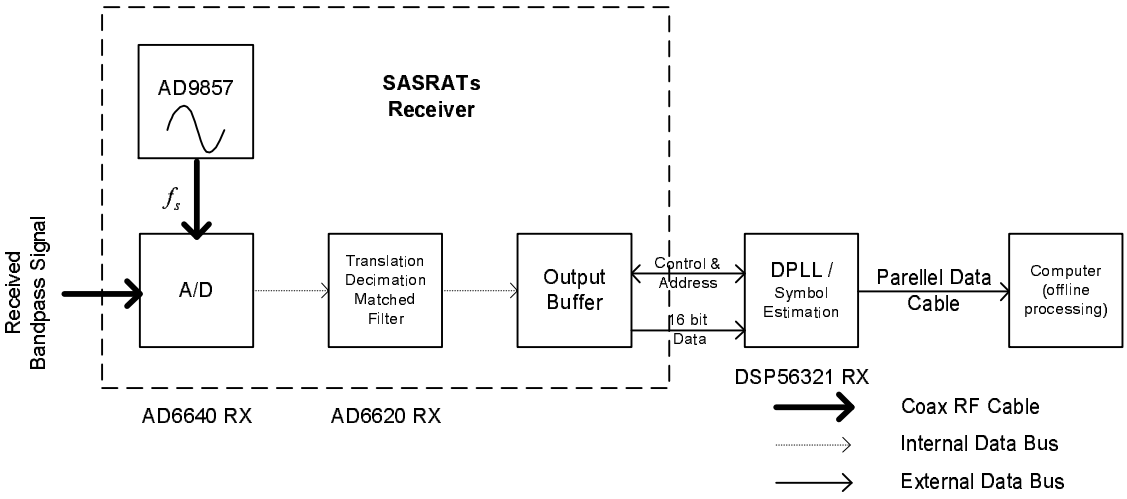

Figure 5.12: Receiver Implementation Utilising SASRATs Hardware

Figure 5.12 illustrates the functional hardware blocks of the receiver and how they interact. The SASRATs receiver is shown with its internal components, and how those components interact with external hardware.

5.5.1 A/D, Bandpass Sampling

The sampling rate was set by programming the AD9857 RX chip as a clock source with a rate of 40MHz. This clock source sets the rate at which the AD6640 RX chip samples the received RF, which has a bandwidth of 250kHz at a carrier frequency of 45MHz.

The spectrum of the digitised signal behaves with characteristics as illustrated in Figure 5.13.

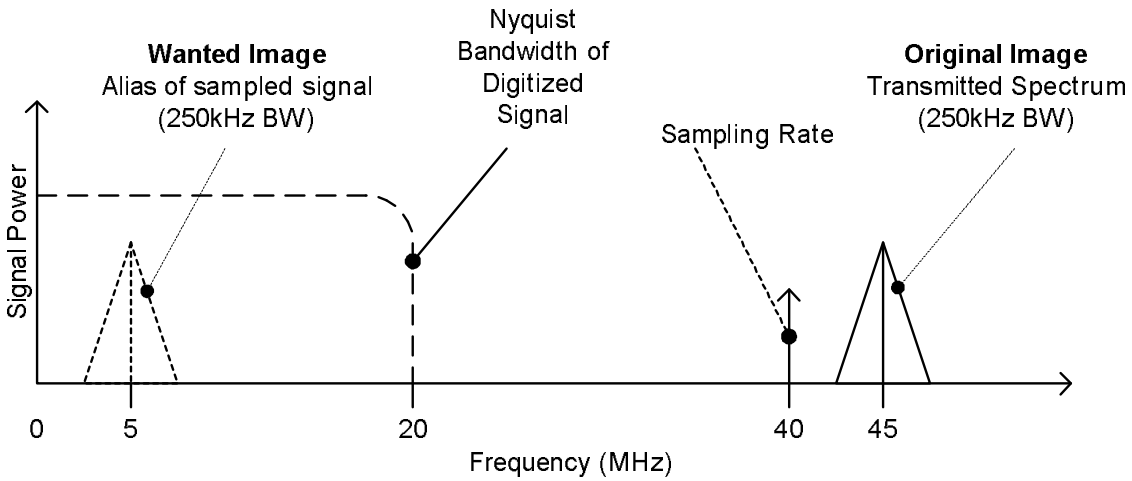

Figure 5.13: Spectral Image Created by Bandpass Sampling of the Received RF Signal (positive frequencies only shown)

Figure 5.13 shows a wanted alias of the spectrum occurring at 5MHz for a signal with a sampling rate of 20MHz.

5.5.2 Frequency Translation, Decimation and Matched Filtering

The second stage in the receiver is performed in the AD6620 RX chip. First the Numerically Controlled Oscillator (NCO) is configured to *translate* or shift in the frequency domain the spectral position of the digital signal. In this case the NCO is configured to translate the spectrum by 5MHz such that the spectrum of the received signal then occurs at baseband.

In the following stages the AD6620 RX decimates the sampling rate and applies a matched filter. Matched filtering occurs at the FIR filter block along with rate decimation as shown in Figure 5.6. The matched filter is configured by AD6620 RX filter design software as depicted in figure 5.14. In this case 160 filter coefficients are computed for a SSRC filter with a rolloff of 0.35. The resultant signal is then a matched-filtered, decimated signal occurring at baseband at a sample rate of 2 samples per symbol, corresponding to 250kHz.

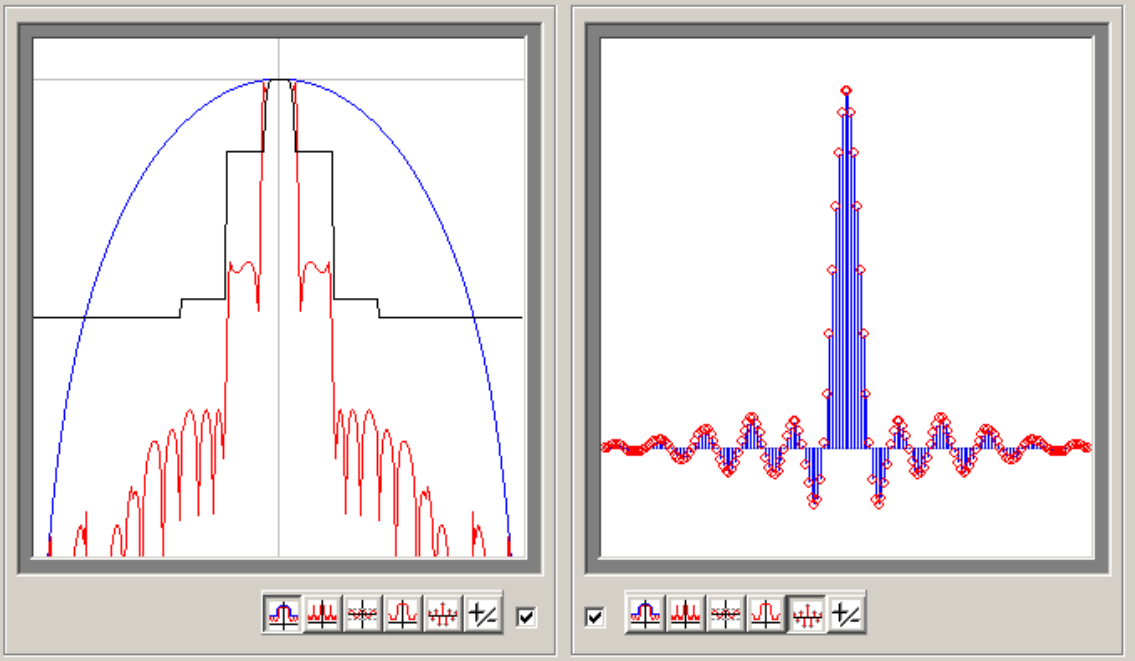

Figure 5.14: Screenshot from AD6620 filter design software; illustrating the design response of the receive filter, spectral mask and impulse response for 160 tap FIR filter.

5.5.3 FFML1 Based DPLL

The FFML1 based DPLL is implemented in software within the DSP56321 RX board which is configured to communicate with the output bus of the SASRATs receiver by means of an external data bus. The DPLL reads incoming samples as the SASRATs control bus indicates they become available. The DPLL software then processes the samples, makes a soft estimate of each incoming symbol and stores those estimates in on-chip DSP memory. The DPLL loop was configured to run a set number of times. In most cases 10,000 symbol estimates were recorded.

5.5.4 Offline Processing

A computer was used to post-process the received symbol estimates offline in order to produce performance data and graphs. Motorola software was used to communicate with the DSP56321 RX board via parallel cable and download symbol estimates from DSP memory to a computer dumpfiles. MatLab software was written to load the dumpfiles, decode from D-QPSK, to calculate BER and also to produce the corresponding calculated phase error.

5.6 Receiver Performance Results

Results are now presented in this section along with an assessment of the DPLL in terms of its ability to acquire phase, track phase and the accuracy of the soft output estimation.

Comparison will be made for varying levels of SNR in an AWGN channel and for varying levels of Doppler frequency in a fading channel. In the case of a fading channel the SNR is held at a constant level of 14dB.

5.6.1 Experimental Testing Procedure

As discussed in section 5.3.2, a known block of 100 binary data bits is encoded into 50 D-QPSK symbols and transmitted in a continual cycle. The DPLL loop, when executed, receives a sequence of 20,000 complex samples. The estimated symbols and their corresponding calculated phase offsets are then dumped to DSP memory for later download to PC. MatLab based scripts are used to post-process the DSP dump file; making decisions on the soft output data, decoding from D-QPSK and calculating an error rate per 50 symbol block of transmitted data.

The timing phase being tracked is both arbitrary and unknown *prior* to execution of the routine, hence the estimated phase curves can appear either positive or negative.

5.6.2 DPLL Bandwidth

In order to determine the maximum operating bandwidth of the DPLL and estimate the effective processing capability additional code was inserted into the DPLL routine that raises a General Purpose Input Output (GPIO) pin on the DSP56321 RX board high at the start of the a single DPLL loop cycle and lower it upon completion. The processing time used was then measured with a high-speed oscilloscope to estimate the timing of the GPIO pin output. This result is seen in table 5.1;

DPLL Version (Interpolator Used)	Execution Time
Linear	920ns
Cubic	945ns

Table 5.1: DPLL Execution Time

The difference in execution time for the two DPLL versions is somewhat less than was anticipated given the additional processing demands required by cubic interpolation compared to linear interpolation.

An execution time of 945ns indicates an upper limit on the symbol-processing rate to be in the order of $10^6 Symbols/s$ or $2MB/s$. The main limiting factors are the rates at which data is read from the receiver, the buffer speed of the receiver and the efficiency with which the data input service interrupt routine is constructed. It could be anticipated that when the DSP56321 RX chip is incorporated with the receiver on board that the efficiency with which the AD6620 RX chip and DPLL communicate could be somewhat improved over the case where communication is implemented by means of external parallel data cables.

5.6.3 Linear and Cubic Interpolator Comparison

Following is a series of plots generated in order to compare the acquisition and in-lock performance of DPLL design based on both linear and cubic interpolator design, and hence determine the the most appropriate DPLL version for a more in depth analysis.

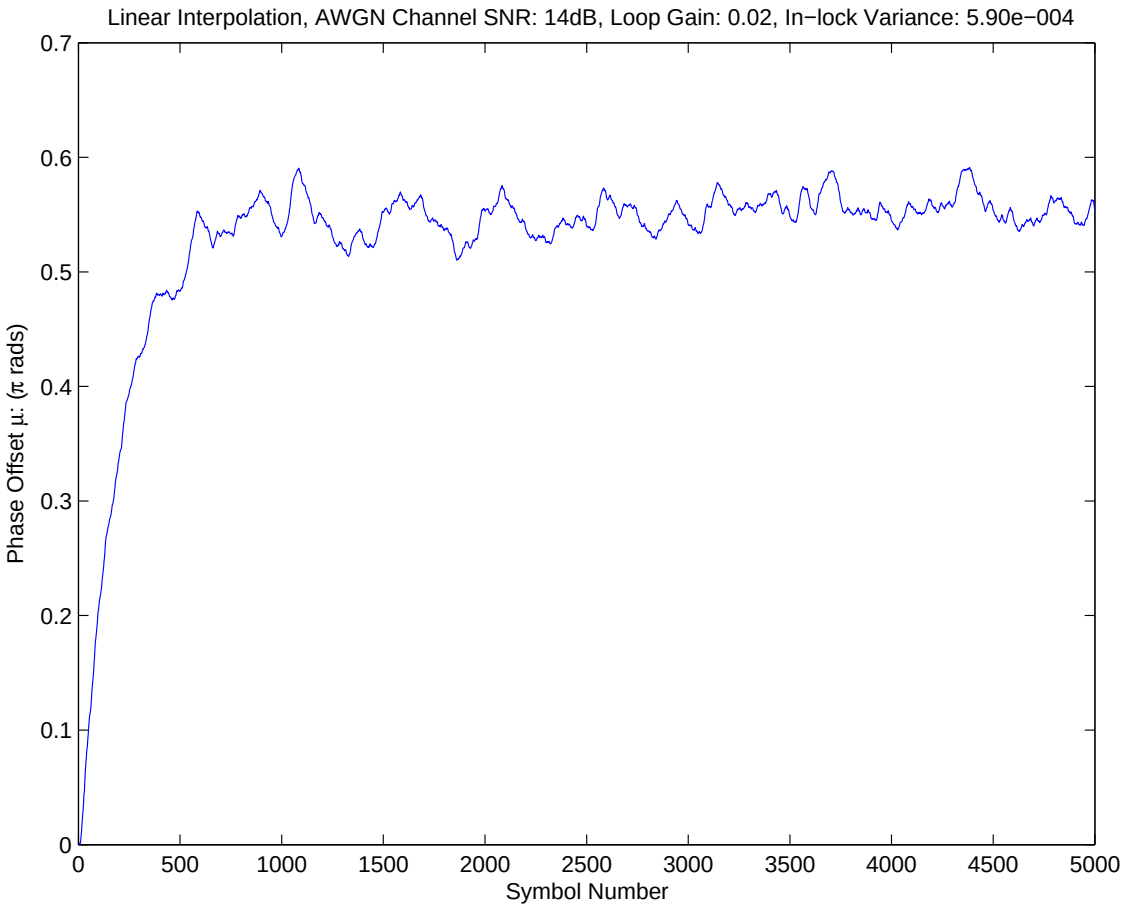

Figure 5.15: Linear Interpolator Based DPLL with a Loop Gain of 0.02 Operating in a AWGN Channel of 14dB SNR

Figure 5.15 shows the acquisition of tracking of the first 5000 symbols for a DPLL utilising linear interpolation. Phase lock is achieved after about 500 symbols.

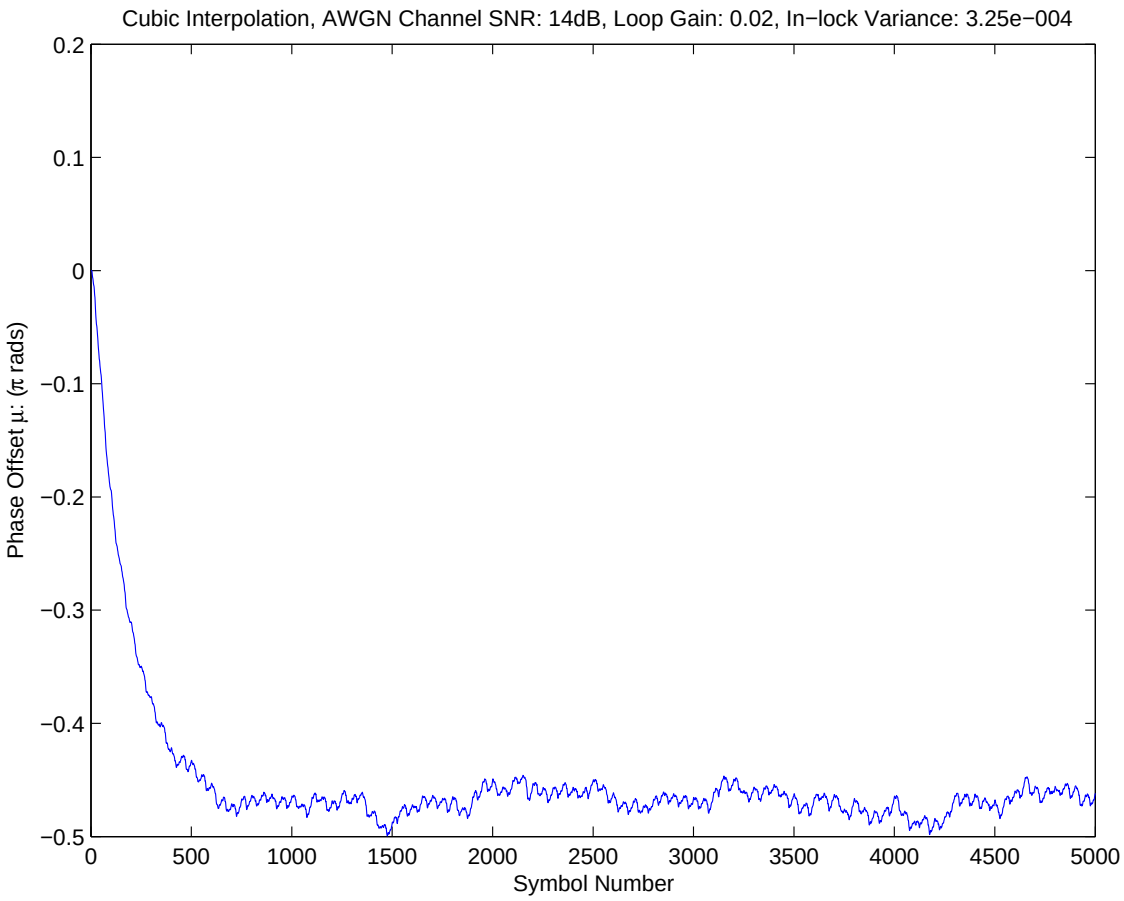

Figure 5.16: Cubic Interpolator Based DPLL with a Loop Gain of 0.02 Operating in a AWGN Channel of 14dB SNR

Figure 5.16 shows the performance for a cubic interpolator based DPLL operating in the same environment as the linear interpolator whose phase acquisition and in-lock performance is illustrated by figure 5.15. Phase lock for both DPLL versions is achieved after about 500 symbols, the differentiating factor being the variance of the in-lock phase estimation.

The next two plots illustrate the change in the acquisition time and in-lock performance for both of the DPLL versions when the loop gain is increased to 0.05 for the same channel.

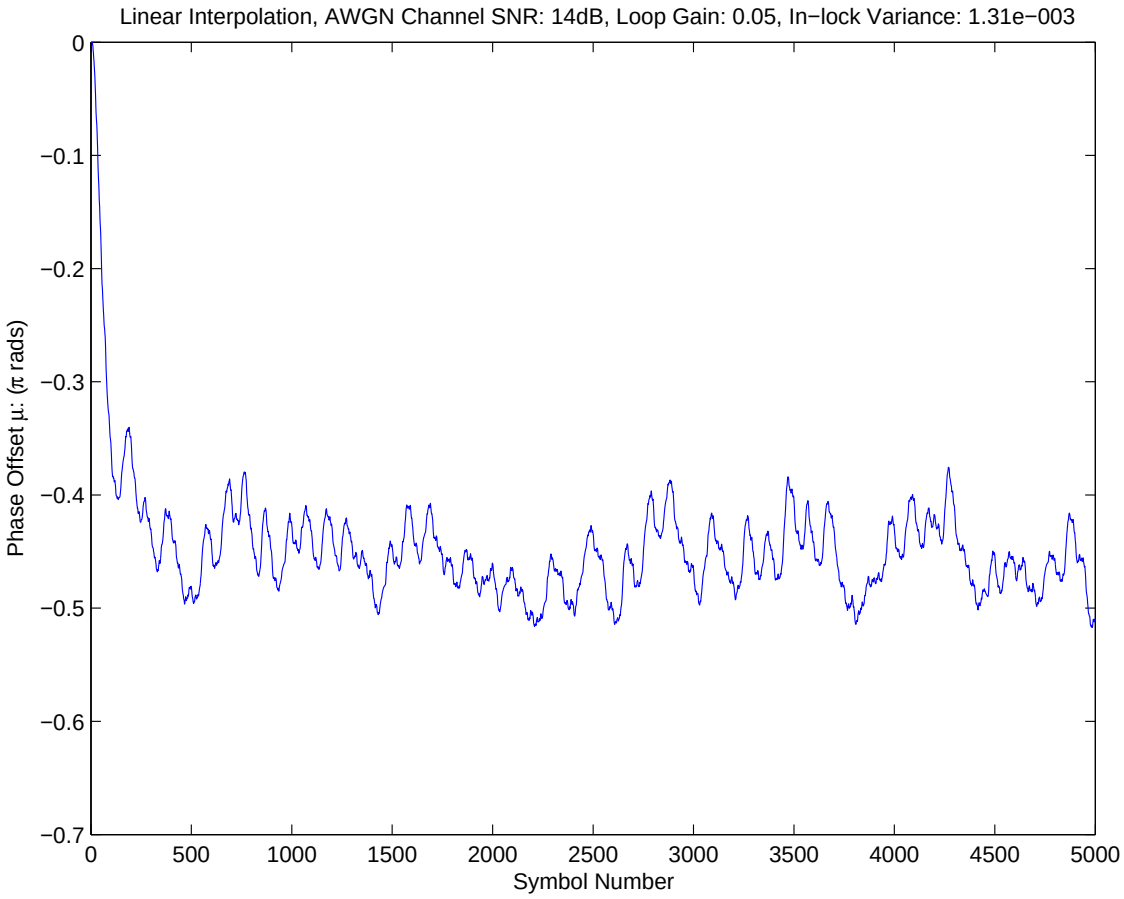

Figure 5.17: Linear Interpolator Based DPLL with a Loop Gain of 0.05 Operating in a AWGN Channel of 14dB SNR

Increasing the loop gain from 0.02 to 0.05 improves the phase acquisition from about 500 symbols (figure 5.15) to a lock time of about 200 symbols, but comes at a cost of increasing the in-lock variance from 5.90e-4 to 1.31e-3.

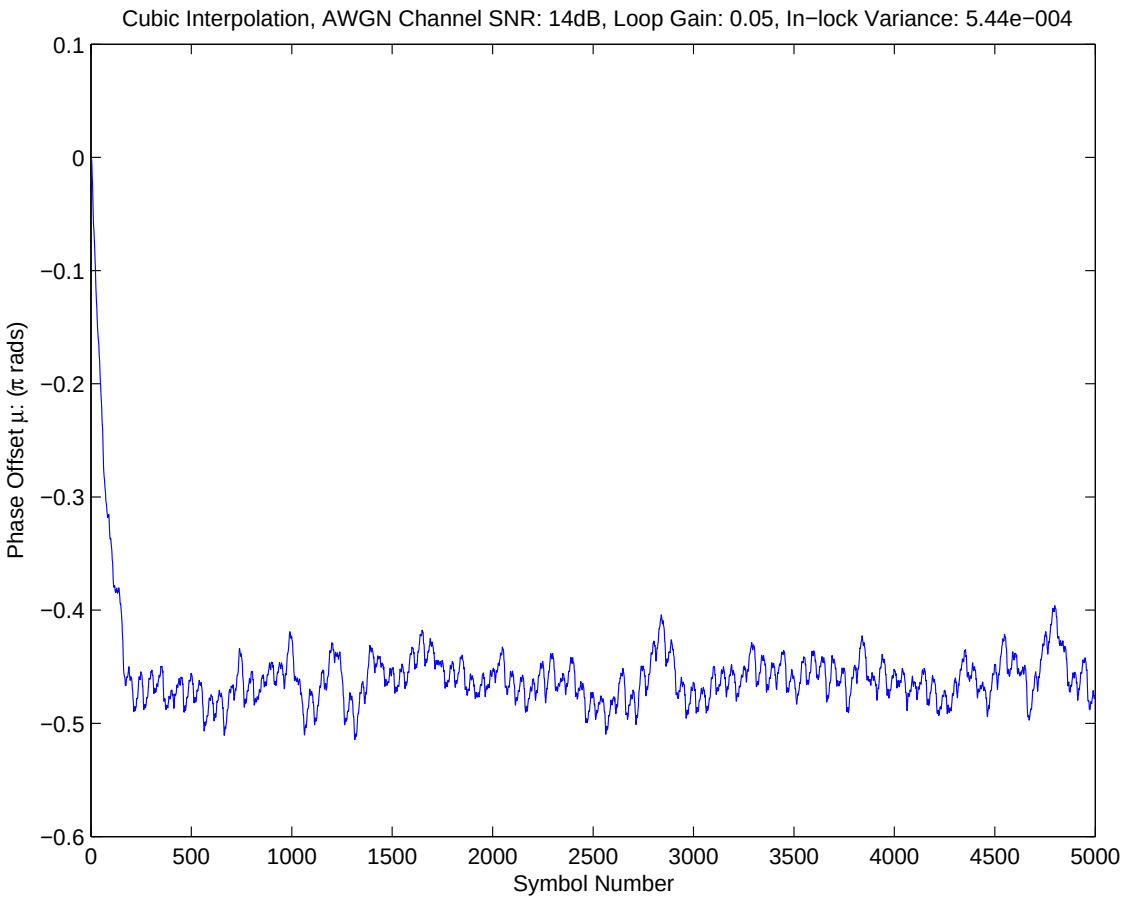

Figure 5.18: Cubic Interpolator Based DPLL with a Loop Gain of 0.05 Operating in a AWGN Channel of 14dB SNR

Figures 5.15-5.18 show some operational differences between a DPLL based on a linear interpolator and a DPLL based on a cubic interpolator. Typically cubic interpolation will produce a more accurate estimation of the phase error. In both cases above the cubic interpolator based DPLL acquires the phase slightly faster and performs better in terms of in-lock variance. Cubic interpolation is calculated at the same sample rate as that of linear interpolation; 2 samples per symbol, but uses 4 samples in each interpolation.

Given that the cubic interpolator only slightly increases the execution time of the DPLL loop (section 5.6.2), for the purposes of the remainder of this thesis the performance metrics of an operational DPLL will be calculated based on a DPLL utilising cubic interpolation. Loop gains above 0.05 are considered impractical, as variance becomes the dominant factor. For the purposes of this thesis a lock period of 200 samples is considered to be adequate.

5.6.4 Phase Acquisition

In the following section comparison is made for the speed of phase acquisition for a given SNR level of 22dB and varying loop gains. This is intended to develop a clear relationship between the loop gain, acquisition time and in-lock performance of the cubic interpolator based DPLL.

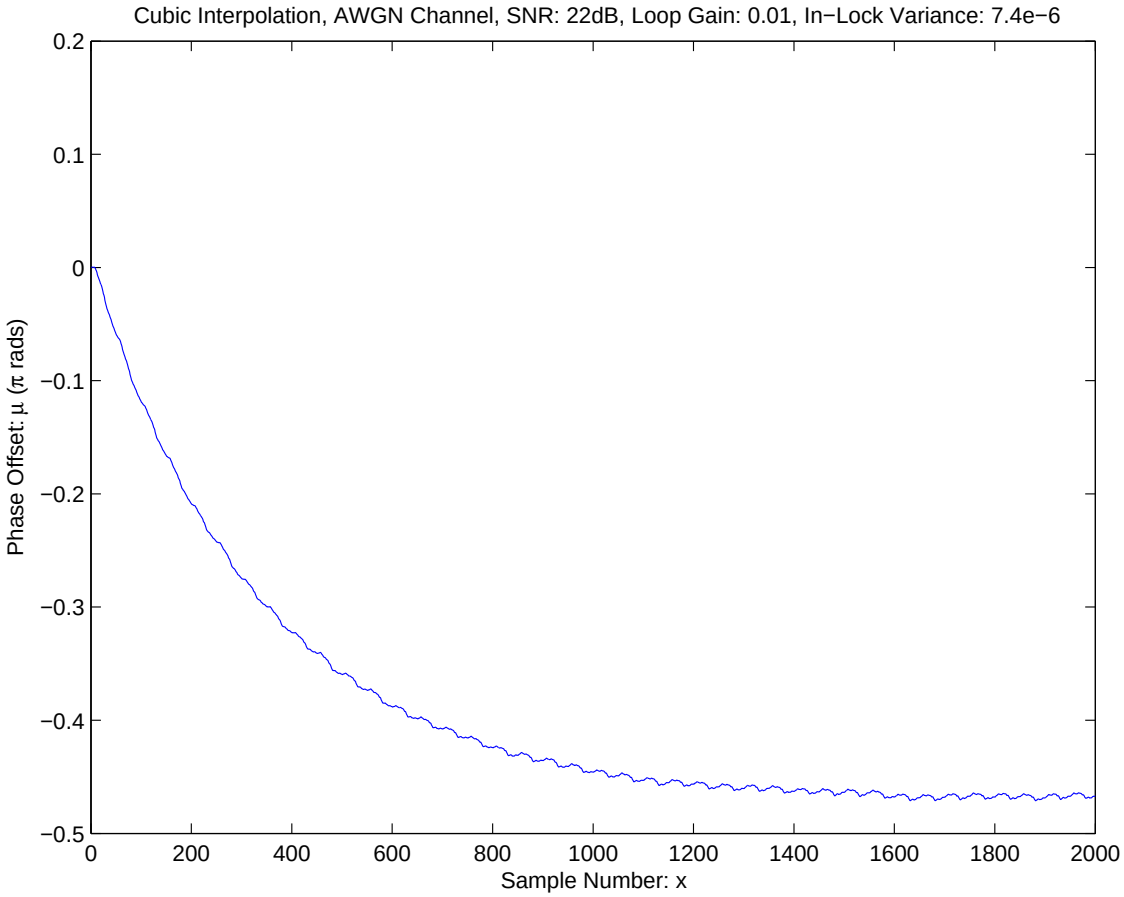

Figure 5.19: Phase Acquisition: Cubic Interpolator Based DPLL with a Loop Gain of 0.01 Operating in a AWGN Channel of 22dB SNR

The plot of estimated phase in figure 5.19 shows a gradual acquisition of the phase error, in this case *phase lock* is achieved after approximately 1500 samples have been received and processed.

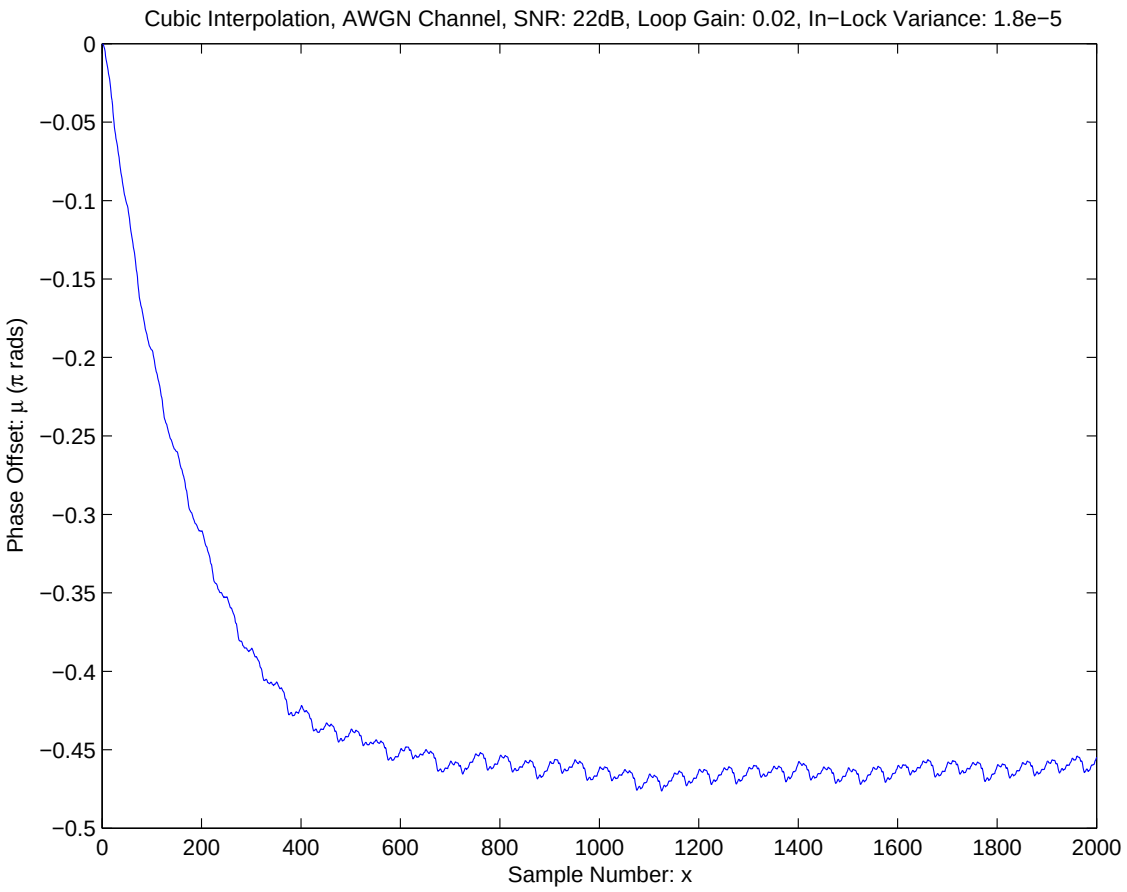

Figure 5.20: Phase Acquisition: Cubic Interpolator Based DPLL with a Loop Gain of 0.02 Operating in a AWGN Channel of 22dB SNR

Doubling the gain of the loop equates to a lock time of approximately half. In this case lock is achieved after approximately 800 samples, the tradeoff being in an increased variance of the in-lock phase error estimation of about an order of magnitude from the previous plot.

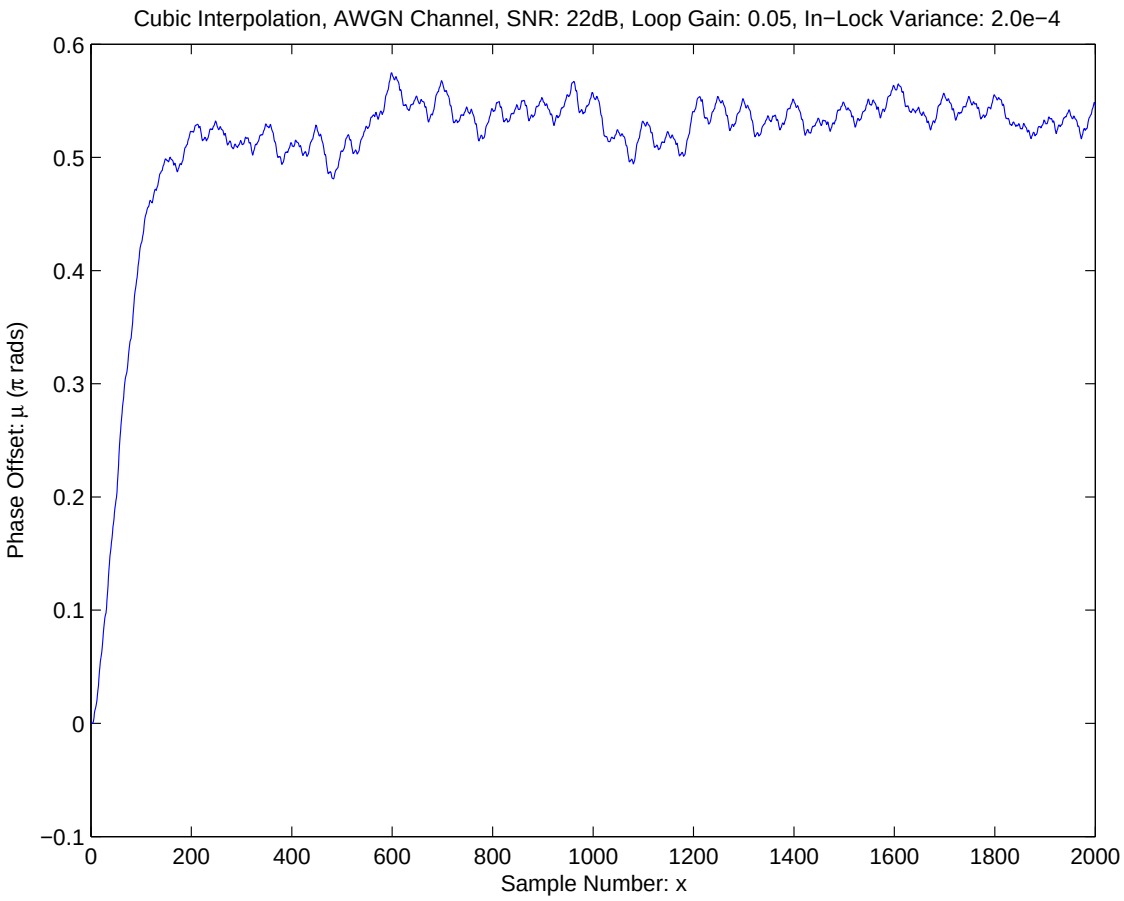

Figure 5.21: Phase Acquisition: Cubic Interpolator Based DPLL with a Loop Gain of 0.05 Operating in a AWGN Channel of 22dB SNR

Figure 5.21 illustrates the acquisition and tracking performance using a loop gain of 0.05. A lock is achieved after approximately 200 samples and the variance of the in-lock phase estimation has increased to 2.0^{-4}.

The table below tabulates approximate lock times related to loop gain values.

Loop Gain	Samples to Lock	Variance
0.01	1500	7.4e-6
0.02	800	1.8e-5
0.05	200	2.0e-4

Table 5.2: Acquisition Times for Loop Gain Values

Table 5.2 indicates a relationship between the number of samples required to achieve a signal lock and the in-lock variance as governed by the loop gain of the DPLL. This general relationship holds for both cubic and linear interpolator versions of the DPLL,

with the cubic version outperforming the linear version in terms of a lower level of in-lock variance by about an order of magnitude.

5.6.5 Decoding Accuracy

For a tangible measure of how accurate the soft output of the DPLL is, the BER rates have been calculated for a range of DPLL loop gain values for a fixed channel SNR of approximately 5dB.

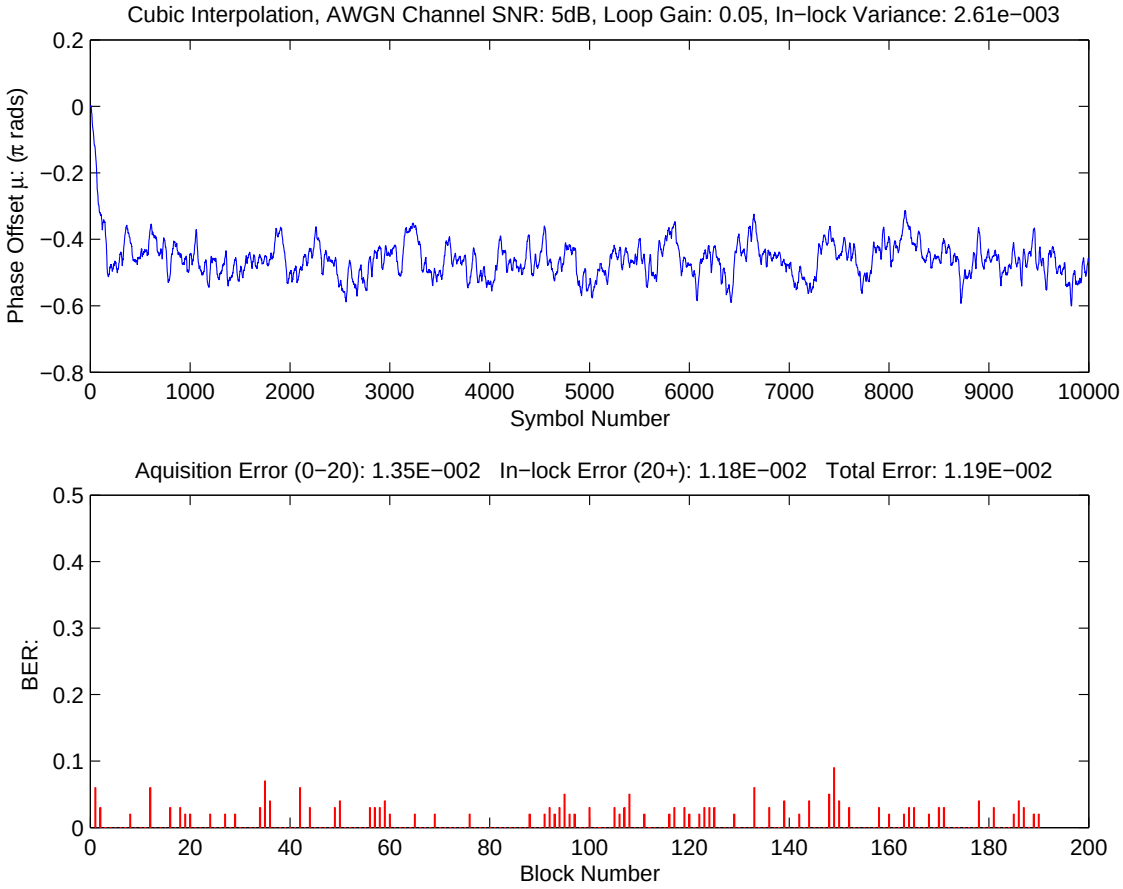

Figure 5.22: Cubic interpolator operating in an AWGN channel of 5dB SNR and a loop gain of 0.05. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

Figure 5.22 shows error ratios for consecutive blocks of 100 bits for a loop gain of 0.05 and a AWGN channel with an SNR of approximately 5dB. The signal is locked onto in a reasonably short period, approximately 200 symbols, but with a high level of

variance for the estimated phase offset. An overall BER of 1.19×10^{-2} is achieved[3]. The theoretical BER for a DQPSK system transmitting over a 5dB SNR channel is 3.04×10^{-2} [37] and for transmission over an AWGN channel with 8dB SNR the theoretical BER is 3.64×10^{-3}.

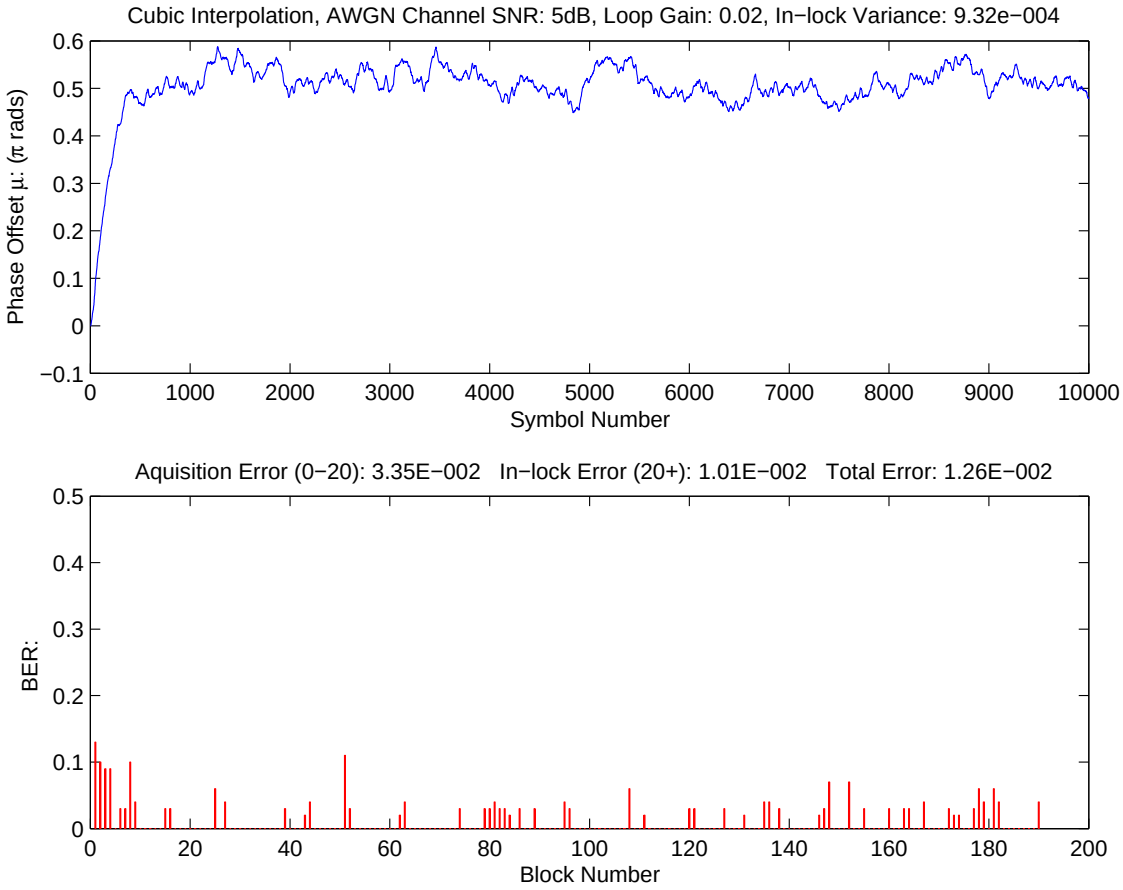

Figure 5.23: Cubic interpolator operating in an AWGN channel of 5dB SNR and a loop gain of 0.02. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

Figure 5.23 illustrates block error ratios for a loop gain of 0.02, while the in-lock variance is an order of magnitude lower, the overall BER is very similar to the error performance of figure 5.22.

[3] The receiver implementation employed in this chapter employs matched-filtering, effectively improving the SNR incident at the decision making point of the receiver by 3dB [7].

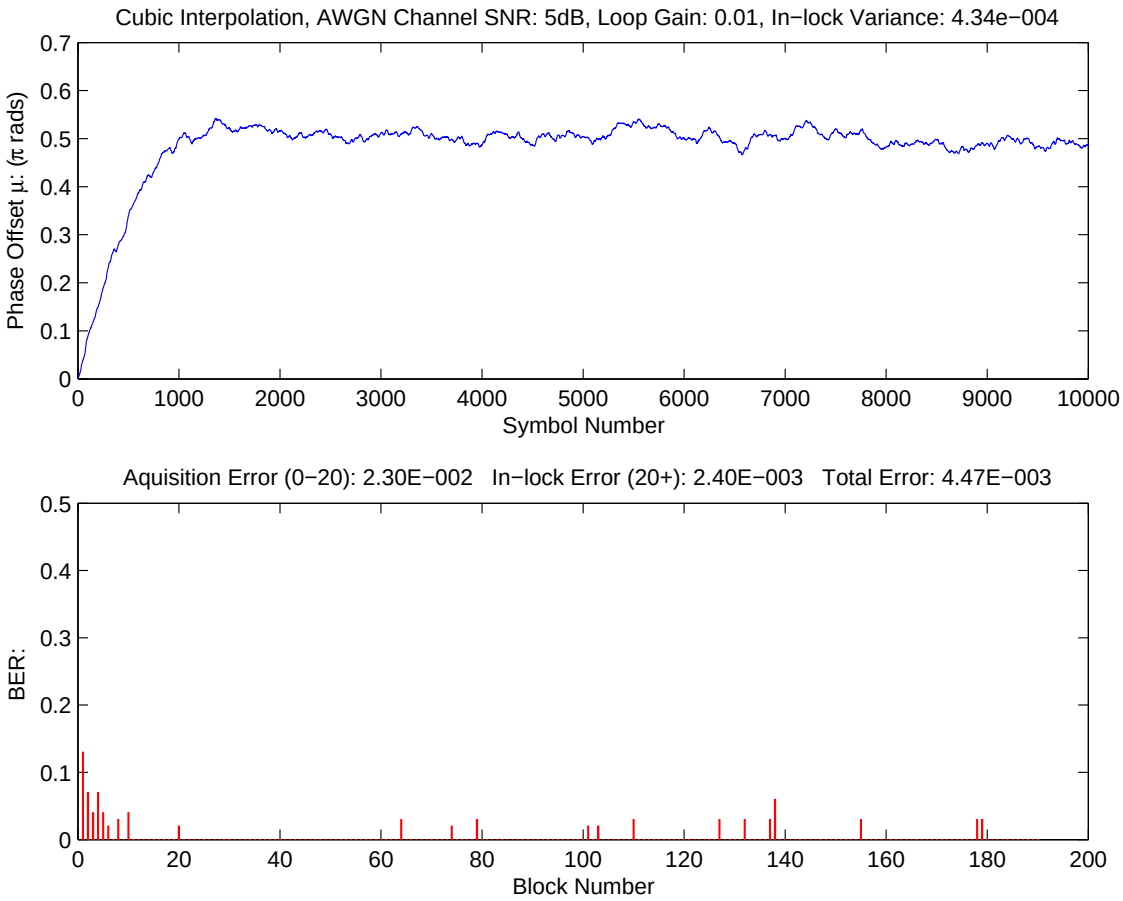

Figure 5.24: Cubic interpolator operating in an AWGN channel of 5dB SNR and a loop gain of 0.01. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

When the loop gain is lowered to 0.01, the error performance is degraded by over a factor of a $\frac{1}{2}$.What also becomes apparent as illustrated in figure 5.24 is the relationship between estimated phase error and the BER per block, in the first ten blocks of the BER plot of figure 5.24 the BER falls rapidly from about 0.12 down to zero as in the same period the phase error is acquired.

The collated variances versus BERs are presented in Table 5.3;

Loop Gain	Variance	In-Lock BER
0.05	2.6e-3	0.012
0.02	9.3e-4	0.010
0.01	4.3e-4	0.045

Table 5.3: Collated Variances and BER's from Figures 5.22, 5.23 and 5.24

Table 5.3 provides an indication of the relationship between loop gain, variance and

BER. When this is considered in conjunction with the results of Table 5.2 the trade-off between increased loop gain providing a shorter acquisition time and decreased loop gain providing an improved in-lock BER performance becomes apparent.

5.6.6 Fading Channel

A fading channel was simulated by utilising an HP 11759B Fading channel simulator. The following two figures present the estimated timing phase error and BER performance of the DPLL for different Doppler fade rates. As discussed in section 5.4.2, there was also an issue with the power level on input to the channel simulator that resulted in degraded receiver performance.

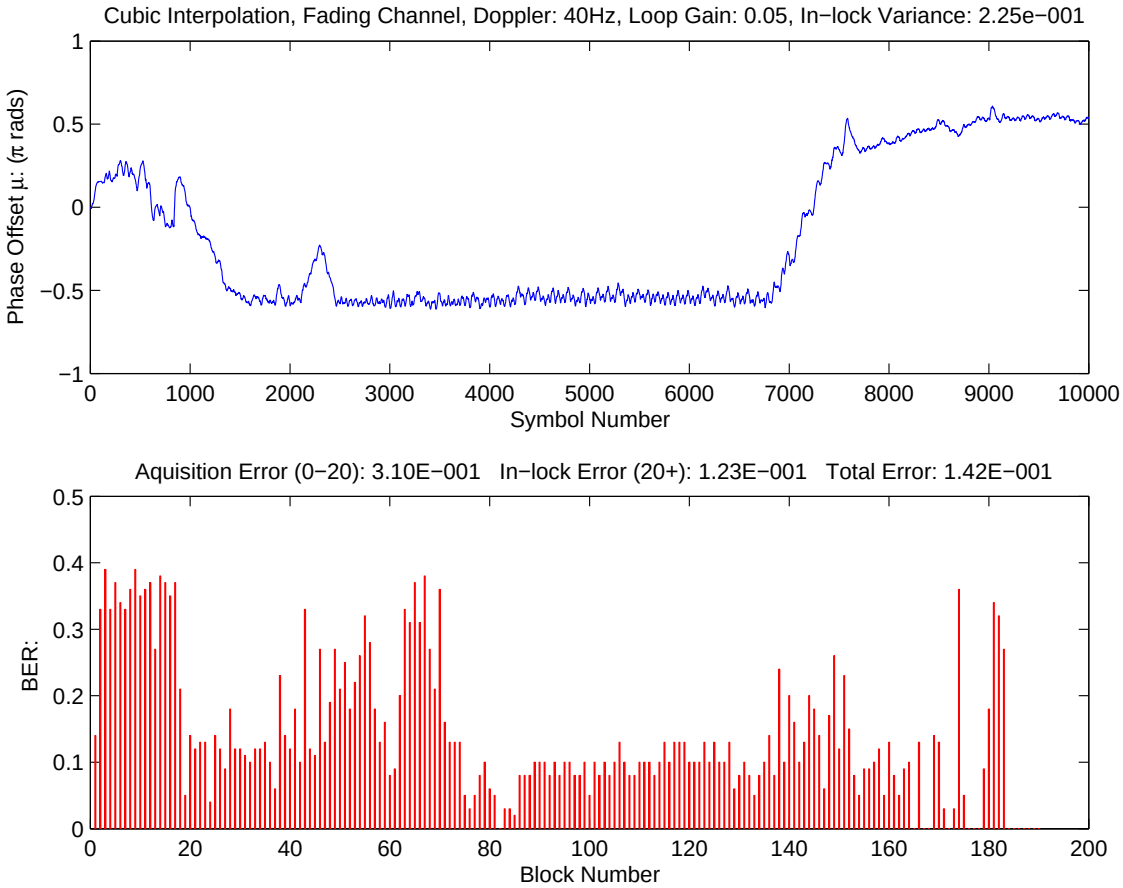

Figure 5.25: Cubic interpolator operating in an flat fading channel with a Doppler frequency of 40Hz and a loop gain of 0.05. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

Figure 5.25 shows that the phase error of the received signal is being tracked after

approximately 1000 symbols are received with the BER indicating that approximately 85-90% of symbols are being correctly decoded. A phase change of the received signal occurs after about 7000 received symbols and the DPLL tracks this phase change with the BER fluctuating with a higher variance but a similar average.

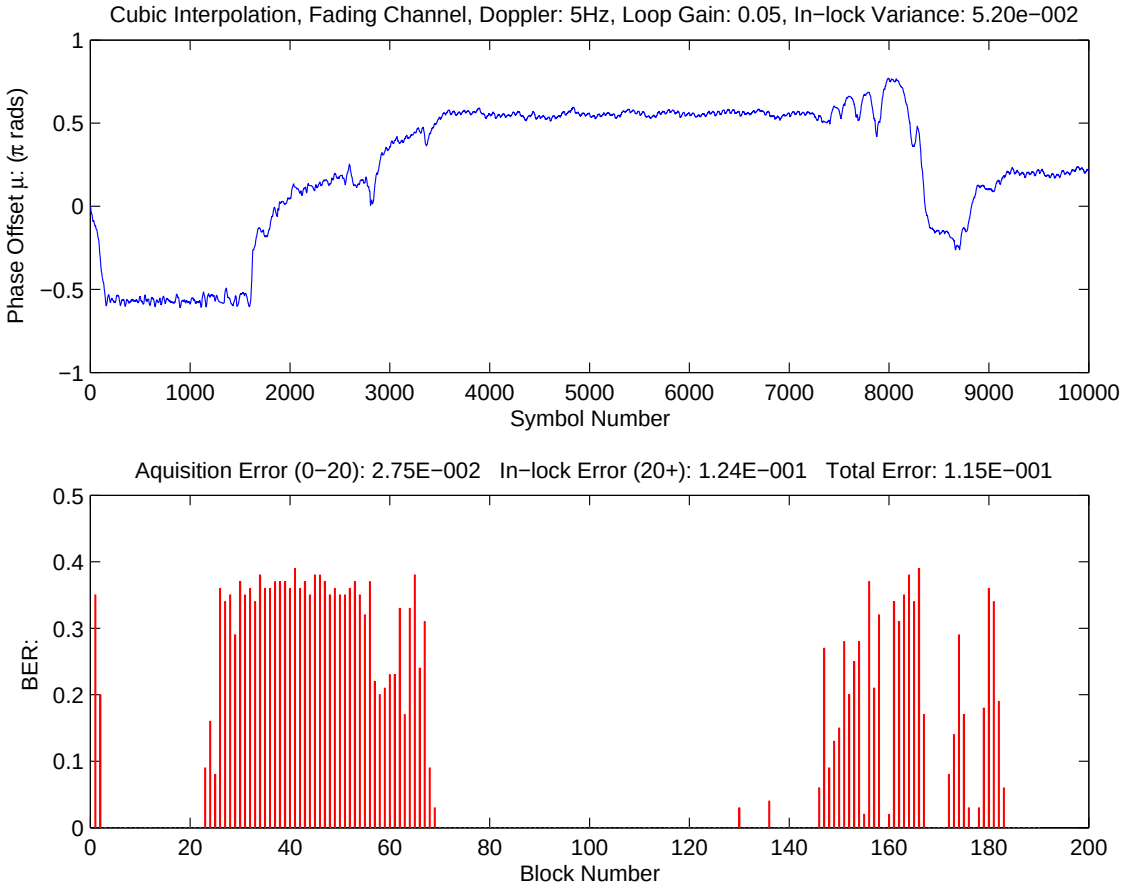

Figure 5.26: Cubic interpolator operating in an flat fading channel with a Doppler frequency of 5Hz and a loop gain of 0.05. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

Figure 5.26 illustrates how at a lower fade rate effects the individual fades become more discernible, in the BER curve. The received phase is tracked with a better level of stability than that in figure 5.25,

5.6.7 Frequency Error Effects

In most practical real-time systems the clock sources for the transmitter and receiver cannot either be perfectly phase synchronised or at *exactly* the same frequency, so the perfor-

mance of the DPLL where there is an error between these clocks must be considered.

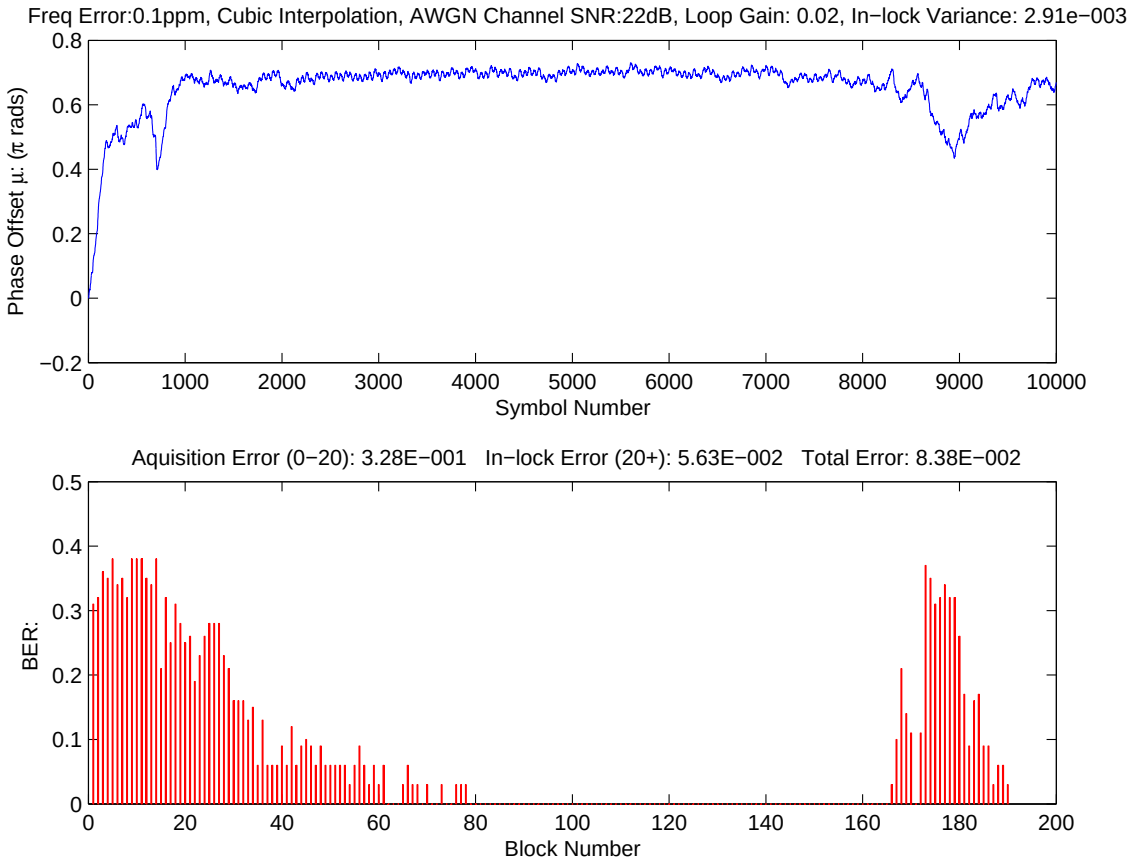

Figure 5.27: Illustrating the effect of clock error between transmitter and receiver of 0.1ppm, for a cubic interpolator operating in an AWGN channel of 22dB SNR and a loop gain of 0.05. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

Figure 5.27 illustrates the signal being acquired within approximately 1000 symbols, with a BER of zero being achieved after approximately 4000 symbols. The phase then continues to be tracked with relative stability, but suffers a burst of errors that occurs at about the 8500^{th} symbol. It is possible that this error burst is a consequence of utilising interpolation at a sampling rate of two samples per symbol; where the phase error changes at some constant rate, there will come a point at which there is a *phase wrap-around* that occurs about the same point where the symbol phase is estimate to occur.

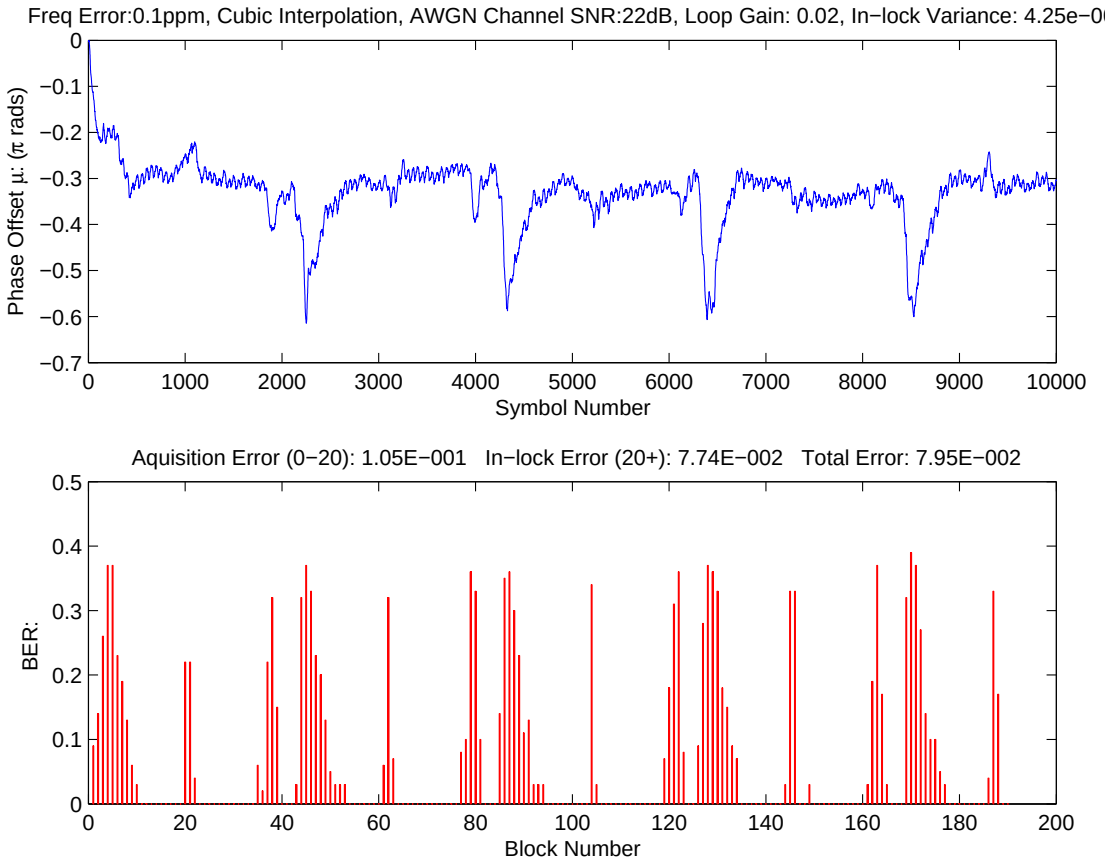

Figure 5.28: Illustrating the effect of clock error between transmitter and receiver of 1ppm, for a cubic interpolator operating in an AWGN channel of 22dB SNR and a loop gain of 0.05. The percentage of errored bits per block of bits (100 bits per block) is illustrated in the lower plot.

When the frequency error is increased as illustrated in figure 5.28, the periodic nature of the error bursts becomes more apparent.

5.7 Summary

In this chapter the structure of the experimental test platform has been presented along with the experimental methodology used to evaluate the performance of the FFML1 based DPLL. Detail was also given of the software-based functional blocks of the DPLL.

Operational results produced in this chapter show the ability of a DSP implemented FFML1 based DPLL to acquire and track the phase of a D-QPSK modulated signal in real time, with two complex samples per symbol at a rate of 125k symbols/second. It is anticipated from operational results that the upper limit of the DPLL implemented on a

56321 DSP board would be in the vicinity of 10^6 symbols per second.

Analysis was undertaken to measure the performance of the receiver in terms of acquisition rate and decoding accuracy where the signal is corrupted by varying levels of AWGN or fading in the communications channel.

The ability of a real-time implementation of an FFML1 DPLL to operate in a channel suffering AWGN or flat fading corruption was demonstrated.

Chapter 6

Conclusions

6.1 Summary

A Digital Phased Locked Loop (DPLL) was implemented on a Motorola 56321 Digital Signal Procesor (DSP) using 56300 Assembly language. The Timing Error Detection algorithm implemented was FFML1, a maximum likelihood based algorithm [8] developed at Canterbury University. This DPLL implementation was successfully integrated with the SASRATS system [5]. Other functional blocks implemented and incorporated into the DPLL were the AGC block, interpolation, gain, first order IIR filter, fade detection and integration block.

The system operated with a quarternary phase shift keying (QPSK) constellation, which as a single level constellation allowed FFML1 to be simplified in implementation and avoided the necessity for the DSP to execute a divide operation. *Differential coding* and demodulation was used, such that the system could then operate without the requirement for absolute phase information to be recovered by the receiver.

A digital implementation of an automatic gain control function was implemented, taking advantage of the integrated DSP co-processor. The implementation of the AGC allowed the DPLL to process sampled data transmitted directly from the SASRATS receiver.

The timing phase acquisition and tracking process was implemented entirely in the digital domain, with a *free running* sampling clock, demonstrating the use of *interpolation* to estimate the symbol at optimal timing point rather than actual sampling at that point.

Two different methods of interpolation were implemented to investigate the trade off between cubic interpolation and linear interpolation techniques. Linear interpolation shows promise with the ability to achieve an acceptable level of symbol estimation accuracy at a reduction of the processing demand required by the cubic interpolator implementation.

To further increase the symbol processing ability, the DPLL was developed to process signal samples at a reduced rate of 2 Samples / Symbol, rather than four as in earlier investigations into FFML1 implementation. The reduction in the requirement to service input requests significantly reduced the execution time of the DPLL loop. It was demonstrated that acquisition and tracking performance of the FFML1 based DPLL was of a level that could be useful in a real-time implementation.

Implementations of the DPLL were *blind*, in that the phase of the received signal was acquired and tracked without any prior knowledge of transmitted symbols. It was demonstrated that prior knowledge of transmitted symbols (a training sequence) would only be of significant value to DPLL acquisition performance when the phase error approached the maximum difference of $\frac{1}{2}\pi$. The FFML1 based DPLL demonstrated the ability to operate blind.

The processing rate of the DPLL was sufficient to demonstrate that the design could be of some practical use. The DPLL also demonstrated the ability to process received data corrupted by both additive white Gaussian noise and flat fading. The effects of symbol frequency error were also investigated, with the DPLL demonstrating the ability to recover phase lock after a symbol overflow or underflow.

In summary, an all-software FFML1 based DPLL was successfully implemented and integrated with a SASRATS receiver. The receiver operated with bandpass sampling, supplying a digitised received signal, transposed to baseband at a rate of two Samples per Symbol. The DPLL acquired and tracked the phase of the received signal, blind and in cases where either Additive White Gaussian Noise or flat fading had been applied. The DPLL demonstrated the ability to process incoming data at a rate of up to 1.2MBs^{-1}.

6.2 Suggestions for Future Work

This implementation of FFML1 restricted the system to transmission of QPSK, given that many digital transmission systems routinely employ constellations of up 32QAM and 64QAM this would seem restrictive. FFML1 in particular was designed with the ability to process multilevel constellations as a fundamental requirement.

One DPLL system bottleneck restricting the symbol processing limit is as at the interface between the SASRATS receiver and the Motorola 56321 DSP. Sixteen bit data was received by the DSP servicing external interrupts. These interupts consumed a significant amount of processing time and increased the DPLL loop execution time. More efficient interfacing should increase the maximum symbol processing rate.

Bibliography

[1] M. Calabrese, "The future of spectrum policy and the future of spectrum policy and the fcc spectrum policy task force report," March 6, 2003.

[2] W. Technologies, "Software defined radio," tech. rep., Wipro, August August 2002.

[3] I. Mitola, J. and G. J. Maguire, "Cognitive radio: making software radios more personal," in *Personal Communications*, vol. vol.6, pp. pp.13–18, IEEE, Aug 1999.

[4] D. Taylor and G. Watkins, "Ml symbol timing recovery in the rayleigh flat-fading channel using baud rate sampling," in *Conf. Proceedings*, pp. pp. 62–65, International Symposium on Signals, Systems, and Electronics (ISSE'01), Tokyo Japan, 24-27 July 2001.

[5] P.J.Green, *Space-time processing: an experimental test platform and algorithms*. Phd thesis, Canterbury University, 2002.

[6] J. M. Wozencraft and I. M. Jacobs, *Principles of communication engineering*. N.Y.: Wiley, 1965.

[7] S. S. Haykin, *Communication systems*. New York: Wiley, 3rd ed ed., 1994.

[8] G. Watkins, "Maximum likelihood symbol timing recovery for digital wireless communication," Master's thesis, Canterbury University, 1997.

[9] J.G.Proakis, *Digital communications*. Boston: McGraw-Hill, 4th ed ed., 2001.

[10] L. Lo Presti, "Fir design of raised-cosine filters," in *Electrotechnics, 1988. Conference Proceedings on Area Communication, EUROCON 88., 8th European Conference on*, pp. 146 – 149, Dept. of Electron., Polytech. of Torino, Italy, 13-17 June 1988.

[11] L.Cordesses, "Direct digital synthesis: A tool for periodic wave generation (part 1)," pp. pp. 50–54, Sig. Proc., IEEE, 2004.

[12] A. Devices, *AD9857 CMOS 200 MSPS 14-Bit Quadrature Digital Upconverter*. Analog Devices, 2004.

[13] R. G. Vaughan, N. L. Scott, and D. R. White, "The theory of bandpass sampling," p. pp. 1973, Sig. Proc., IEEE, vol 3 1991.

[14] J. Doble, *Introduction to radio propagation for fixed and mobile communications*. Boston: Artech House, 1996.

[15] M. J. B. Lee, "Comparison between path-loss prediction models for wireless telecommunication system design," in *Antennas and Propagation Society International Symposium*, vol. 2, pp. 186 – 189, Dept. of Radio and Broadcasting Syst. Eng., Kyunghee Univ., Suwon, South Korea, IEEE, 8-13 July 2001.

[16] M. Hata, "Empirical formula for propagation loss in land mobile radio services," pp. pp. 317–325, Trans on Vehicular Technology, IEEE, vol. VT-29 1980.

[17] B. Sklar, "Rayleigh fading channels in mobile digital communication systems.i. characterization," in *Communications Magazine*, p. pp.90, IEEE, vol.35 1997.

[18] W. C. Y. Lee, *Mobile cellular telecommunications systems*. New York: McGraw-Hill, 1989.

[19] H. H. Hmimy and S. C. Gupta, *Statistical model of delay spread and coherence bandwidth for wide-band PCS at millimeter-waves in an urban mobile radio environment*. 1996.

[20] August 2006.

[21] D. D. Gregory, *Space-time wireless channels*. Upper Saddle River, N.J.; London: Prentice Hall, 2002.

[22] R. Clarke, "On synchronisation issues in wireless mobile digital communications," Master's thesis, University of Canterbury, 2002.

[23] F. M. Gardiner, “Interpolation in digital modems-part 1: Fundamentals,” in *Comms. Trans.*, pp. pp. 501–507, IEEE, March 1993.

[24] J. Costas, “Synchronous communications,” in *Communications*, pp. 99–105, IEEE Transactions, Volume 5, Issue 1, March 1957.

[25] M. Ellis, “Using mixers in radio communications.” http://members.tripod.com/michaelgellis/mixerscom.html, 1999.

[26] R. Schafer and L. Rabiner, “A digital signal approach to interpolation,” in *Proc.*, pp. pp. 692–701, IEEE, IEEE, June 1973.

[27] L. Erup, F. M. Gardiner, and R. A. Harris, “Interpolation in digital modems-part ii: Implementation and performance,” in *Trans. on Comms.*, pp. pp. 998–1008, IEEE, IEEE, June 1993.

[28] C. Farrow, “A continuously variable digital delay element,” in *Int. Symp. Circuits. Systs. (ISCAS-99)*, pp. pp. 2641–2645, IEEE. Proc., June 1988.

[29] G. Watkins, “Optimal farrow coefficients for symbol timing recoery,” in *IEEE Trans. Comms. Letters*, pp. 381–383, IEEE, September 2001.

[30] K. H. Mueller and M. Muller, “Timing recovery in digital synchronous data receivers,” in *IEEE Trans. Comms.*, pp. 516–531, IEEE, May 1976.

[31] F. M. Gardner, *Phaselock techniques*. New York: Wiley, 2nd ed ed., 1979.

[32] F. M. Gardiner, “A bpsk/qpsk timing error detector for sampled receivers,” in *IEEE Transaction on Communications*, pp. pp 423–429, vol. COM-34 1986.

[33] D. Verdin and T. C. Tozer, “Symbol-timing recovery for m-psk modulation schemes using the signum function,” in *Presented at IEE Colloquium on New Synchronisation Systems for Radio Systems*, (London,UK), 1995.

[34] Motorola, *DSP56321RM/D - DSP56321 Reference Manual*. Motorola, 2001.

[35] A. Devices, *AD6640 12-Bit, 65 MSPS IF Sampling A/D Converter*. Analog Devices, 2003.

[36] A. Devices, *AD6620 67 MSPS Digital Receive Signal Processor*. Analog Devices, 2001.

[37] G. Ferrari and G. Corazza, "Tight bounds and accurate approximations for dqpsk transmission bit error rate," in *IEE, ELECTRONICS LETTERS*, vol. Vol. 40, 30th September 2004.

Appendix A

Simulation and Experimental Procedure

Two types of experimental simulations were conducted to produce the results presented throughout this thesis; *Off-line simulation* and *Real-time simulation*. The method of each of these each of these simulations is presented in the following sub-sections.

A.1 Hardware Configuration

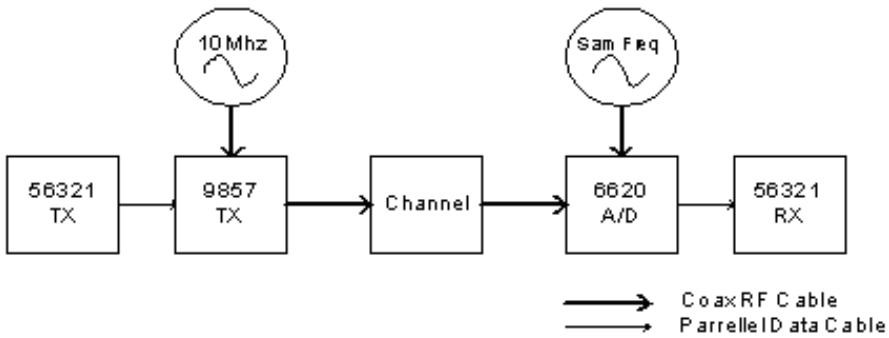

Figure A.1: Hardware Setup Overview

56321TX Data source, symbol encoding and pulse shaping function; `QPSK_TXT.cld` was executed, running a loop that repeatedly delivers the 9857TX chip a known block of complex symbols, 4 times oversampled and pulse shaped with a square root raised consine pulse with a rolloff of 0.35

9857TX Performed Direct Digital Synthesis to produce a QPSK modulated signal about a specified carrier. The below figures illustrate setting that were used to produce 125k Symbols/second about a carrier of 45MHz;

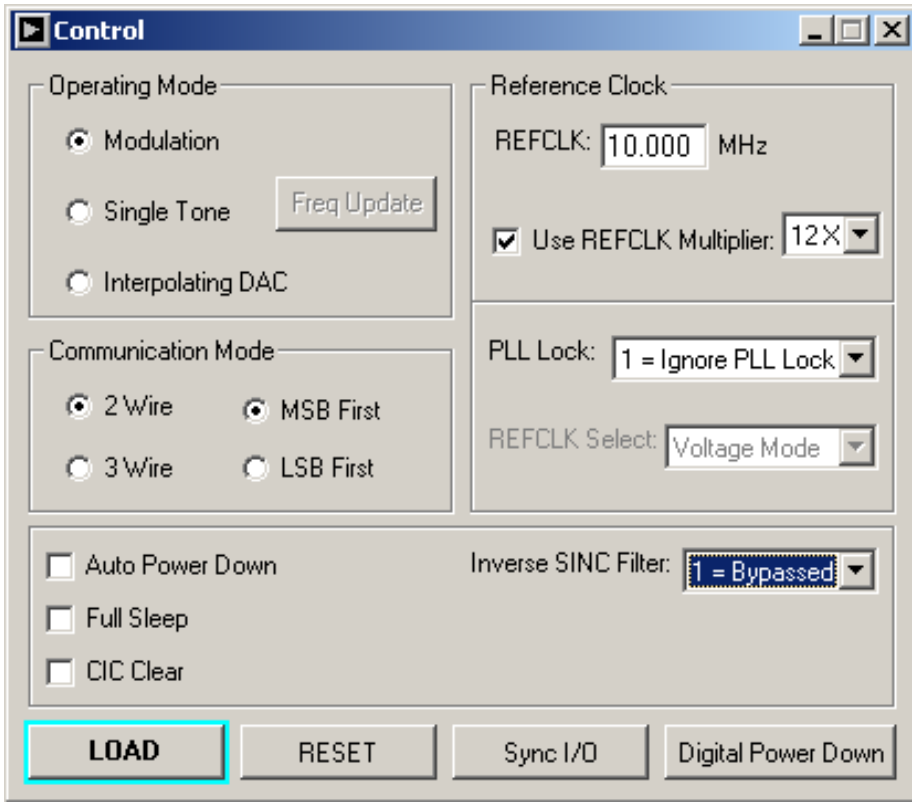

Figure A.2: First page of 9857TX Chip Settings

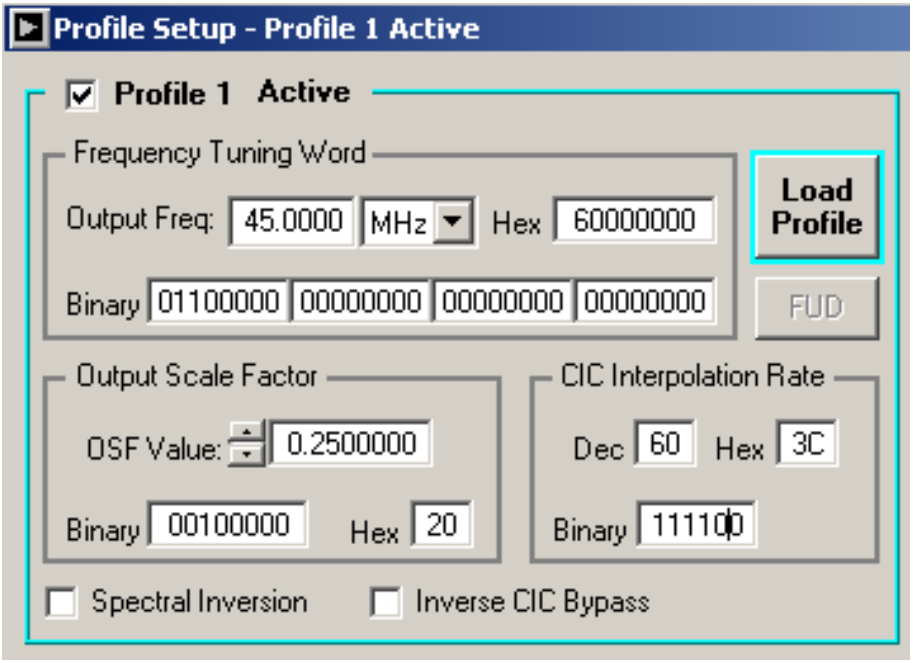

Figure A.3: Second page of 9857TX Chip Settings

Channel Simulated using either an attenuation block to reduce SNR to simulate AWGN or the HP11759B Fading channel simulator

6620A/D Set to perform bandpass sampling on the received QPSK carrier signal. Sampling clock was to a rate of 40MHz, the digital signal then SRRC matched filter & decimation down to baseband, at sample rate of 2 Samples per Symbol, 250k Symbols per second, as shown below;

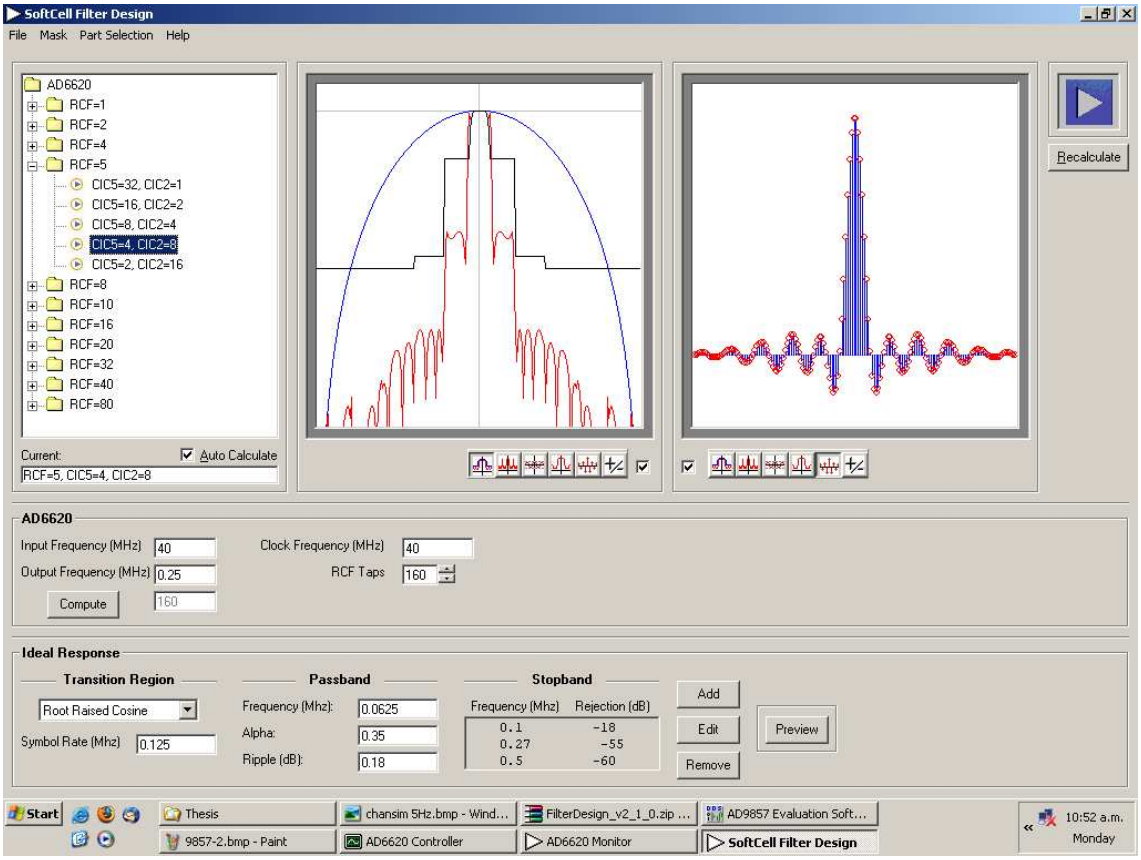

Figure A.4: Screenshot showing settings as loaded into the 6620A/D chip

56321RX Performs the AGC, DPLL and produces a soft symbol estimate. DSP software file `DPLL-Main-Cubic.cld` or `DPLL-Main-Linear.cld` was executed to producing soft output and phase estimation for 10,000 Symbols (dumped to DSP memory and then imported by MatLab routine)

A.2 Hardware tools

Configuration of the signal processing chips used was performed using a PC loaded with appropriate software to interface with hardware.

For interface with the 56321 DSP, the assembler, disassembler and monitor software[1] had the below interface;

[1] `DSP56300_Tools.exe` from *Metrowerks*

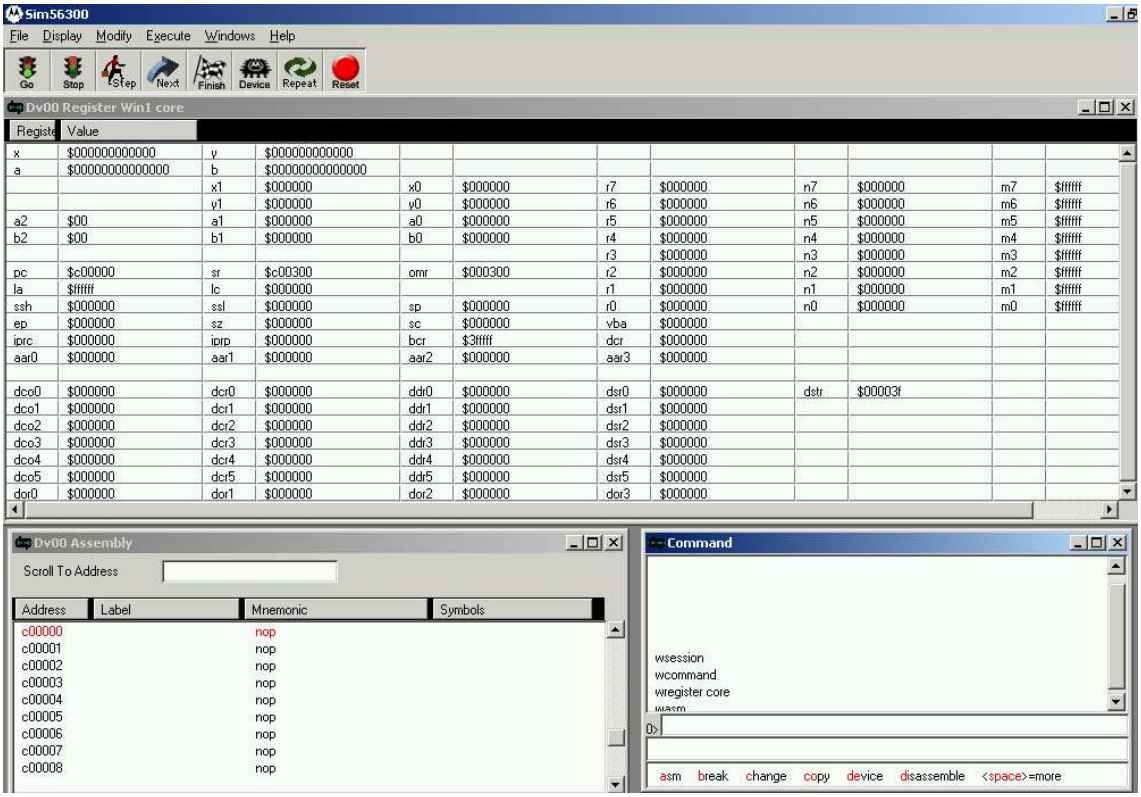

Figure A.5: Screenshot of the Assembler, Disassembler and Monitor used

The 'Command' window in the lower right hand corner can be clicked on and used to enter commands directly to the monitor interface, this command interface is utilised in section A.4.

A.3 Software Configuration

A fixed directory structure was used, as illustrated below;

Figure A.6: Simulation Directory Structure

The directory was used to store assembly code files (ASM) compiled into object files that could be directly loaded into the 56321 DSP, and MatLab files, used for both preprocessing and post-processing.

The files are listed and described in the following two subsections

A.3.1 DPLL ASM Files

This set of files are compiled together to create the object file for the DPLL. the `DPLL-Main-XXXX` file incorporates code to include other files as required for the object code build. There is a set of these files for each of the linear and cubic interpolator versions of the software, also for real-time and offline versions. Stored in directories;

`\DPLL-Final\ASM\Cubic\Offline`

`\DPLL-Final\ASM\Cubic\Realtime`

`\DPLL-Final\ASM\Linear\Offline`

`\DPLL-Final\ASM\Linear\Realtime`

`DPLL-Main-Cubic.asm` Digital Phase Locked Loop code for cubic interpolator version

DPLL-Main-Linear.asm Digital Phase Locked Loop code for linear interpolator version

DPLL-mem-map.asm Sets out the memory map for the program, declares locations of all program variables

DPLL-interupts.asm Interrupt routines for reading in external sample data and processing EFCOP output to determine AGC parameters

DPLL-equates.asm Defines program equates

IOEQU.asm Standard 56300 equates

A.3.2 TX ASM Files

This set of files incorporates software loaded into the 56321TX chip that provides a source of known data, encodes it into QPSK symbols and produces four times oversampled SRRC pulse shaped baseband I and Q waveforms. These files are stored in directory;

`\DPLL-Final\ASM\Transmitter`

QPSK_TXT.asm Main file compiled to generate complete object code for I and Q signal source. Encodes data bits into symbols, pads with zeros for oversampling and passes I and Q signals through FIR filter for SRRC pulse shaping

TXT-Frame-bits.asm Data file with block of bits to be transmitted; this file created by MatLab script `CreateDataFrame.m`, stored in directory `\DPLL-Final\DataFiles`

TXT-RC_pulse_16T_4sps_p35RO.asm Data file with the the coefficients of the pulse shaping FIR filter. This file generated by `CreateSRRCPulse.m`, stored in directory `\DPLL-Final\DataFiles`

A.3.3 MatLab Files

These are the script files used for preprocessing to create data files and post processing for the conversion, analysis and display of DPLL system performance. These files are stored in directory;

`\DPLL-Final\Matlab`

`CreateDataFrame.m` Creates 100 bits of random data, saves to `FRAME-Data_bits.mat` and encodes to 2's complement fractional numbers and saves to `TXT-FRAME-bits.asm`

`CreateSimRecData.m` Creates simulated received signal samples (for use with offline simulation). Uses data in `FRAME-Data_bits.mat` to produce 2 samples per symbol of user configurable corruption (AWGN or flat fading) with a specified phase offset. Saves simulated received signal to `Frame-Rec_Sig.lod` for direct loading into DSP

`CreateSRRCPulse.m` Creates a set of coefficients for pulse shaping by FIR filter process in 56321TX. Saves coefficients in file `TXT-RC_pulse_16T_4sps_p35RO.asm`

`ImportDataFrames.m` Used in realtime simulations. Imports dump files from DPLL DSP: `DSP-Mu_out.lod` and `DSP-Soft_out.lod`. Decodes data from soft symbol estimates, calculates BER and produces plot of estimated phase, BER

`ImportDataFramesSim.m` Used for offline DPLL simulations. Imports dump files from DPLL DSP: `DSP-Mu_out.lod` and `DSP-Soft_out.lod`. Decodes data from soft symbol estimates, calculates BER and produces plot of estimated phase, BER

`Enc_DQPSK.c` Subroutine for encoding bits to DQPSK symbols. Written in 'c' and MEX compiled for speed

`Dec_DQPSK.c` Subroutine for decoding DQPSK symbols to bits. Written in 'c' and MEX compiled for speed

`DSP_Imp.c` Subroutine for converting 2's complement fractional hexadecimal numbers into decimal. Used to convert dumpfiles from DSP. Written in 'c' and MEX compiled for speed

`create_fading.m` Subroutine for creating a simulated fading channel

`h2d.m` Subroutine for converting a 2's complement fractional hexadecimal number into decimal

ImportDSP.m Subroutine for importing DSP series data

ImportDSP2.m Subroutine for importing DSP interleaved data

writedbit.m Write a byte of data in DSP 'lod' format for export to DSP

A.4 Simulation Process

Following is the process used for the simulation of DPLL. The hardware is configured as per section A.1 and the software as per section A.3.

1. Create block of random data and coefficients for pulse-shaping (required to be run only once)

   ```
   CreateDataFrame.m
   CreateSRRCPulse.m
   ```

2. Execute transmitter code

   ```
   load "ASM\Transmitter\QPSK_TXT.asm"
   go
   ```

3. Run DPLL in DSP: Execute the commands from the monitor

   ```
   force s
   load "DataFiles\FRAME-Rec_Sig.lod"
   ```

   Run the appropriate DPLL code, one of

   ```
   load "ASM\Cubic\Offline\DPLL-Main-Linear.cld"
   load "ASM\Cubic\Offline\DPLL-Main-Cubic.cld"
   load "ASM\Cubic\Realtime\DPLL-Main-Linear.cld"
   load "ASM\Cubic\Realtime\DPLL-Main-Cubic.cld"
   ```

   followed by

   ```
   go
   ```

wait until loop execution is complete, then

```
save y:1000..3710 “DataFiles\DSP-Mu_out.lod”
save y:4000..8e20 “DataFiles\DSP-Softout.lod”
log off c
```

4. Then execute MatLab script:

```
ImportDataFrames.m
```

Appendix B

DPLL Assembly Code

B.1 DPLL-MAIN-CUBIC.ASM

```
        include 'ioequ.asm'
        include 'DPLL-equates.asm'
        include 'DPLL-mem-map.asm'
        include 'DPLL-interrupts.asm'
        include 'DPLL-initialisation.asm'

;*********************************************************************************
; Main Program loop
;*********************************************************************************

        org     p:$400
MainLoop

        move    #$2710,x0             ; 10,000 symbols
        DO      x0,ENDLOOP
        ;DO      forever,ENDLOOP

; Read two complex samples

;        jclr    #0,x:DFlag,*
        jsr     s_IRQA

        jset    #0,x:wrap,AGC
;        jclr    #0,x:DFlag,*
        jsr     s_IRQA

        jclr    #1,x:wrap,AGC
;        jclr    #0,x:DFlag,*
        jsr     s_IRQA

; Service AGC (EFCOP) Routine if data ready

AGC     jsset   #15,y:M_FCSR,AGCOut

;  Cubic Interpolator

Mu_Def  move    x:mu,x0                               ; Build Mu matrix => [mu^3 mu^2 mu 1]
        mpy     x0,x0,a
        nop
        move    a,y0
```

```
        mpy     x0,y0,a  a1,x:mu2                       ;mu^2 -> Mu(1,2)
        nop
        move    a1,x:mu3                                ;mu^3 -> Mu(1,1)
;                                                       ;Begin Matrix multiplication
        nop
        nop
        bset    #M_SM,sr


MuXA    move            x:(r3)+,x0  y:(r5)+,y0
        mpy     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        nop
        move    a1,y:(r6)+
        mpy     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        nop
        move    a1,y:(r6)+
        mpy     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        nop
        move    a1,y:(r6)+
        mpy     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a  x:(r3)+,x0  y:(r5)+,y0
        mac     x0,y0,a
        nop
        move    a1,y:(r6)+

        move            x:(r1)+,x0  y:(r6),y0
        mpy     x0,y0,a  x:(r1)+,x0  y:(r6)+,y0
        mpy     x0,y0,b  x:(r1)+,x0  y:(r6),y0
        mac     x0,y0,a  x:(r1)+,x0  y:(r6)+,y0
        mac     x0,y0,b  x:(r1)+,x0  y:(r6),y0
        mac     x0,y0,a  x:(r1)+,x0  y:(r6)+,y0
        mac     x0,y0,b  x:(r1)+,x0  y:(r6),y0
        macr    x0,y0,a  x:(r1)+,x0  y:(r6)+,y0
        macr    x0,y0,b

        move    y:M_SO,r7
        nop
        bclr    #M_SM,sr
        move    a1,x:iSig_R
        move    b1,x:iSig_I

        move    a1,y:(r7)+
        move    b1,y:(r7)+

        move    x:PSym,x1
        move    x:NSym,y1
        move    r7,y:M_SO

        brclr   #23,a1,*+4
        move    y1,x:(r0+4)             ;x:(r0+4)
        bra     *+2
        move    x1,x:(r0+4)             ;x:(r0+4)

        brclr   #23,b1,*+4
        move    y1,x:(r0+5)             ;x:(r0+5)
        bra     *+2
        move    x1,x:(r0+5)             ;x:(r0+5)


FFML1
```

```
; FFML1 Timing Error Estimation

; yr*yr+yi*yi -> A_op

A_      move    x:iSig_R,x0                                     ; Yi
        mpy     x0,x0,a         x:iSig_I,x0                     ; Yi^2
        macr    x0,x0,a         x:(r0),x1                       ; Yr^2+Yi^2
        nop
        move    a1,x:(r2)+

; 2*(Ar[k-1]*(Ar[k]-Ar[k-2])+Ai[k-1]*(Ai[k]-Ai[k-2])) ->D_op

D_      move    x:(r0+4),a                                      ;
        sub     x1,a                                            ; Ar[k]-Ar[k-2]
        move    x:(r0+2),x0
        move    a1,y0                                           ;
        mpy     x0,y0,a                                         ; Ar[k-1]*(Ar[k]-Ar[k-2]) -> A
        move    x:(r0+1),x1
        move    x:(r0+5),b                                      ;
        sub     x1,b                                            ; Ai[k]-Ai[k-2]
        move    x:(r0+3),x0
        move    b1,y0                                           ;
        mpy     x0,y0,b         x:B_op,x1                       ; Ai[k-1]*(Ai[k]-Ai[k-2]) -> B
        add     a,b                                             ; A+B

; mu=(A-B)*D

AmBxD   move    x:(r2-2),a
        sub     x1,a              b1,y0                         ; (A_op-B_op) -> a
        nop
        move    a1,x0
        mpy     x0,y0,a           x:Gain,y0

FadeD   clr     b                   #0.125,x1
        move    x:FdThr,x0          a,y1
        rep     #8
        mac     y1,x1,b             x:(r3)+,y1
        cmpm    x0,b                a,x0
        jge     LGain
        move    #0,y0

LGain   mpy     x0,y0,a           x:mu_z,x0

Filter  move    x:Pole,y0
        mac     x0,y0,a           x:I2,b                        ; mu=mu+Pole*muz_z
        nop
        nop

Integ   bclr    #M_L,SR
        add     b,a               a1,x:mu_z                     ;  (a) I1 = I2 + mu
        nop
        add     a,b               a1,x:I2                       ;  (b) mu = I1 + I2  , (I2=I1)
        nop
        nop

WrapD   jclr    #M_L,SR,Save
        brset   #23,x:mu,*+5
        move    #>2,x0
        bra     *+3
        move    #>1,x0
        move    x0,x:wrap

Save    move    y:M_mu_t,r7
        nop
        nop
        move    b1,x:mu
        move    b1,y:(r7)+
        move    r7,y:M_mu_t

        lua     (r0+2),r0                                       ; next symbol
```

```
ENDLOOP

movep #0,x:M_IPRC ; resets the interrupt priority register-core (IPRC)

        debug

        jmp   *

;*************************************************************************
        end
```

B.2 MEM-MAP.ASM

```
;*************************************************************************
;   DPLL routine ASM file. Roger Kippenberger 2005
;*************************************************************************
;
;   Equates file
;
;*************************************************************************


SET_BCR EQU $3fffff ;BCR for 1 wait state; 3fffff-7 wait states; 3febff-2wait states
SET_AAR2 EQU $050835 ;sets interface external address x:$050000
m_idata         EQU     $050000
m_qdata         EQU     $050001


;*************************************************************************
; Co-processor equates
;*************************************************************************

FCON            equ     $1          ; Enable EFCOP mode: FIR, real, single channel
FIR_LEN         equ     256         ; EFCOP FIR length
IFV             equ     0.00390625  ; 1/FIR_LEN
FDCH            equ     $F00        ; Filter decimation / channel count (F00 = 16 Dec, 1 ch)
```

B.3 INITIALISATION.ASM

```
;*************************************************************************
; INITIALISE RESET VECTOR & Data input interrupt
;*************************************************************************

org p:$000 ;start program at location p:0000
jmp START     ;DSP looks for instructions

        org     p:$10           ;IRQA data input vector
        bset    #0,x:DFlag
```

```
        nop

;*************************************************************************
; Co-processor Initialisation - FIR filter used for AGC
;*************************************************************************

        org     p:$100
START

;*************************************************************************
; PLL, Interupts Initialisation
;*************************************************************************

   movep   #$02000e,x:M_PCTL   ;Set up PLL, 14 X PLL & PSTP bit: fast recovery from STOP state
movep #$c1fc0f,x:M_DSCR ;The most important instruction for stable DPLL operation!!
        movec   #0,sp                   ;init stack pointer

IRQA movec #$c00000,SR ; clear interrupt mask bits in Status Register (SR)
movep #$0,x:M_IPRC ; resets the interrupt priority register-core (IPRC)
bset #M_IAL0,x:M_IPRC ; set IRQA low priority bit
bset  #M_IAL1,x:M_IPRC ; set IRQA high priority bit
        bset  #M_IAL2,x:M_IPRC ; set IRQA edge trigger mode

        movep #SET_BCR,x:M_BCR ; Set BCR register to value in SET_BCR
  movep #SET_AAR2,x:M_AAR2 ; Set AAR2 register for addressing at $050000



EFCOP   move    #FDBA,r3                ; FDM (input data buffer) memory area
        move    #0,x0
        rep     #FIR_LEN
        move    x0,x:(r3)+              ; clear FDM memory area

        move    #FCBA,r3                ;
        move    #IFV,x0
        rep     #FIR_LEN
        move    x0,y:(r3)+

        movep   #FIR_LEN-1,y:M_FCNT   ; FIR length
        movep   #FDBA,y:M_FDBA        ; FIR input buffer Start Address
        movep   #FCBA,y:M_FCBA        ; FIR Coeff buffer Start Address
        movep   #FDCH,y:M_FDCH        ; Filter decimation & Channel count
        movep   #FCON,y:M_FCSR        ; Enable EFCOP with FIR, real, multichannel, shared coefficients

;*************************************************************************
; Buffer initialisation
;*************************************************************************

M_buff  move    #0,x0
        move    #Sym,r0                ; Symbol buffer pointer
        move    #5,m0
        rep     #6
        move    x0,x:(r0)+

        move    #Sam,r1                ; Sample buffer: pointer for data being processed in FFML1
        move    #7,m1
        rep     #8
        move    x0,x:(r1)+

        move    #A_op,r2
        move    #7,m2
        rep     #8
        move    x0,x:(r2)+

        move    #SigI,r4
        move    #7,m4
        rep     #8
        move    x0,x:(r4)+

        move    #mu3,r3
        move    #3,m3
```

```
        move    #mat_A,r5
        move    #15,m5

        move    #MuA,r6
        move    #3,m6

        move     #mu_t,r7
        move     r7,y:M_mu_t
        move     #Sam_s,r7
        move     r7,y:M_SI
        move     #SftOut,r7
        move     r7,y:M_SO


;        jmp      MainLoop

;*************************************************************************
; AGC initialisation
;*************************************************************************

Prime_EFCOP
        do      #128,end_p

;        jclr    #0,x:DFlag,*
        jsr     s_IRQA
        jsset   #15,y:M_FCSR,AGCOut
        nop
end_p


        jmp     MainLoop                    ; Start main program loop
```

B.4 INTERUPTS.ASM

```
;*************************************************************************
;   DPLL routine ASM file. Roger Kippenberger 2005
;*************************************************************************
;
;   Main loop, includes 2.5 point Linear interpolation function,
;   FFML1 based TED, Fade detection, gain, filter and wrap-around detection
;
;*************************************************************************

;*************************************************************************
; Data input interrupt  IRQA
;*************************************************************************

        org     p:$200

s_IRQA  bclr    #0,x:DFlag

        move    y:M_SI,r7
        nop

        move    x:m_qdata,a
        move    x:(r7)+,a
        nop
```

```
        tfr     a,b
        abs     b
        nop
        move    b1,y:M_FDIR
        move    x:AGC_lb,b1
        move    x:AGC_sd,x0
        normf   b1,a
        nop
        move    a1,y0
        mpy     x0,y0,a
        nop
        move    a1,x:(r1)+

        move    x:m_idata,a
        rep     #8
        nop
        move    x:m_idata,a
        move    x:(r7)+,a
        nop

        tfr     a,b
        abs     b
        nop
        move    b1,y:M_FDIR
        move    x:AGC_lb,b1
        move    x:AGC_sd,x0
        normf   b1,a                    ; Apply AGC, normalise to MSB,
        nop
        move    a1,y0
        mpy     x0,y0,a                 ; then multiply down
        nop
        move    a1,x:(r1)+

        move    r7,y:M_SI

        rts

;*************************************************************************
; EFCOP output service
;
;       AGC_lb = Leading bits of avg[power]
;       AGC_sd = multiplier to scale down to B_op value abs( B_op / M_FDOR )
;
;*************************************************************************

AGCOut  movep   y:M_FDOR,a

        lsl     a
        clb     a,b
        normf   b1,a

        move    b1,x:AGC_lb

Divn    abs     a
        move    #0.5,b
        move    a1,x0

        rep     #$18
        div     x0,b                                    ; b = Num, x0 = Div

        move    b0,x:AGC_sd
        rts
```

B.5 MEM-MAP.ASM

```
;*****************************************************************************
;
;                       Memory Map
;                      ============
;
;       p:$0       Boot & interrupt vectors
;       p:$100     Initialisation code
;       p:$200     Interrupt service routines
;       p:$400     DPLL loop program runtime code
;
;       x:$0       L:$0 space for storing 1/2 of 48 bit accumuator (paired with y:$0)
;       x:$2       DPLL Variables
;       x:$100     EFCOP Filter Data Base Address (FDBA) - for AGC filter
;       x:$200     DPLL Buffers
;       x:$50000   Port address to read in phase data (AAR2)
;       x:$50001   Port address to read quadrature data (AAR2)
;
;       y:$0       L:$0 space for storing 1/2 of 48 bit accumuator (paired with x:$0)
;       y:$2       EFCOP Filter Cooefficient Base Addrsss (FCBA) - for AGC
;       y:$100     Storage space for calculated TED results (Mu)

;*****************************************************************************
; Variables using short address locations (6 bit addresses)
;*****************************************************************************

        org     x:$0
;A_stor1 dc      0             ; For storing accumulator 'A'
mu3     dc      0                                       ; mu^3  Mu(1,1)
mu2     dc      0                                       ; mu^2  Mu(1,2)
mu      dc      0                                       ; mu    Mu(1,3)
        dc      $7FFFFF                                 ; 1     Mu(1,4)

Gain    dc      0.05
AGC_lb  dc      $0            ; Average received signal power
AGC_sd  dc      $7FFFFF
A2_s    dc      0             ;
x0_s    dc      0
DFlag   dc      0             ; Data input ready flag
mu_z    dc      0             ; delayed 'mu'
I2      dc      0             ; Variable used for integration
B_op    dc      0.0003750787170082l1
D_op    dc      0             ;   ""
iSig_R  dc      0             ; Interpolated signal (Real)
iSig_I  dc      0             ; Interpolated signal (Imaginary)
BCnt    dc      0             ; Counter for samples collected
nthree  dc      -3
four    dc      4
one     dc      $7FFFFF
p25     dc      0.25
two     dc      2
FdThr   dc      0.01
PSym    dc      0.25
NSym    dc      -0.25
zero    dc      0
Pole    dc      0.9
wrap    dc      0

A_op    dsm     8


;*****************************************************************************
; FCOP Filter Data Base Address (FDBA) - for AGC filter
;*****************************************************************************

FDBA    dsm     FIR_LEN
```

```
;*****************************************************************************
; Circular buffers, long address locations
;*****************************************************************************
;*****************************************************************************
; Estimated Symbol buffer 'Sym';
;       uses 'r0' register, m0=5 (3 complex symbols)
;       r0   = Re(a[k-1]), r0+1 = Im(a[k-1])
;       r0+2 = Re(a[k]),   r0+3 = Im(a[k])
;       r0+4 = Re(a[k+1]), r0+5 = Im(a[k+1])
;*****************************************************************************

Sym     dsm     6

;*****************************************************************************
; Input signal sample buffer initialisation
;       uses 'r1' register, m1=7 (4 complex samples)
;       r1   = Re(x[k-1]), r1+1 = Im(x[k-1])
;       r1+2 = Re(x[k]),   r1+3 = Im(x[k])
;       r1+4 = Re(x[k+1]), r1+5 = Im(x[k+1])
;       r1+5 = Re(x[k-2]), r1+6 = Im(x[k-2])
;*****************************************************************************

Sam     dsm     8

;*****************************************************************************
; Delay Buffer for Interpolated signal point
;
;*****************************************************************************

SigI    dsm     8

        org     x:$5000
Sam_s   ds      $8000

;*****************************************************************************
; Variables using short address locations (6 bit addresses)
;*****************************************************************************

        org     y:$0
A_stor2 dc      0
MuA     dsm     4
M_mu_t  dc      0
M_SO    dc      0
M_SI    dc      0

        org     y:$20
mat_A   dc      0.166667                        ;A(1,1)
        dc      0                               ;A(2,1)
        dc      -0.166667                       ;A(3,1)
        dc      0                               ;A(4,1)
        dc      -0.5                             ;A(1,2)
        dc      0.5                              ;A(2,2)
        dc      $7FFFFF                         ;A(3,2)
        dc      0                               ;A(4,2)
        dc      .5                              ;A(1,3)
        dc      $800000                         ;A(2,3)
        dc      -0.5                             ;A(3,3)
        dc      $7FFFFF                         ;A(4,3)
        dc      -0.166667                       ;A(1,4)
        dc      0.5                              ;A(2,4)
        dc      -0.333333                       ;A(3,4)
        dc      0                               ;A(4,4)

;*****************************************************************************
; Averaging filter for EFCOP AGC calculation
;*****************************************************************************

FCBA    dsm     FIR_LEN

        org     y:$1000
```

```
mu_t    ds      $3E00

;*****************************************************************************
; Soft output
;*****************************************************************************

        org     y:$4000
SftOut  ds      $5E20
```

B.6 IOEQU.ASM

```
;*****************************************************************************
;   DPLL routine ASM file. Roger Kippenberger 2005
;*****************************************************************************
;
;   Equates file
;
;*****************************************************************************


SET_BCR EQU $3fffff ;BCR for 1 wait state; 3fffff-7 wait states; 3febff-2wait states
SET_AAR2 EQU $050835 ;sets interface external address x:$050000
m_idata         EQU     $050000
m_qdata         EQU     $050001


;*****************************************************************************
; Co-processor equates
;*****************************************************************************

FCON            equ     $1          ; Enable EFCOP mode: FIR, real, single channel
FIR_LEN         equ     256         ; EFCOP FIR length
IFV             equ     0.00390625  ; 1/FIR_LEN
FDCH            equ     $F00        ; Filter decimation / channel count (F00 = 16 Dec, 1 ch)
```

Appendix C

MatLab Routines

C.1 CreateSimRecData.m

```
%-------------------------------------------------------------------------------------
%   MatLab routine written by Roger Kippenberger, 2005.
%-------------------------------------------------------------------------------------
% CREATE simulated recevied signal;
%       adjust channel variables: SNR, Fading, avg signal power
%
%-------------------------------------------------------------------------------------
%
% INPUT: Data frame file: FRAME-Data_bits.mat, created by
%       'CreateDataFrame.m'
%
% OUTPUT: Simulated received data: 'FRAME-Rec_Sig.lod' for loading into DSP,
%

clear; close all

FrameLength = 100
SNR=15                        % Channel signal to noise
fdT=0.00001                        % Channel fade rate
AvSigLevel = 0.1                 % Average signal level, for testing AGC
rolloff=.35                       % Rolloff coefficient for RC pulse shape
sps=2                           % Output samples per symbol required
ovs=16                            % Symbol oversample rate - for generating phase offset
phase=0.5                         % Phase offset between data and symbols -0.5 < phase < 0.5
TotalSymbols = 10000              % Total number of simulated symbols to generate samples for
FiltLen = 8                       % in Symbol periods

% Load Data Frame

load('..\DataFiles\FRAME-Data_bits.mat')

DataBits = DataFrame

for n=2:1:2*TotalSymbols/FrameLength
    DataBits=cat(1,DataBits,DataFrame)
end

% Encode data to D-QPSK Complex Symbols

EncData = Enc_DQPSK(int32(DataBits))
t = reshape(double(EncData),2,[])
```

```
Symbols    = 0.25*(t(1,:) + t(2,:)*i)
Symbols_TX = (t(1,:)+1)+(t(2,:)+1)/2

% Create oversampled sample series

SymbolSeries = reshape(vertcat(Symbols,zeros(ovs-1,length(Symbols))),1,[])

% Create SRRC pulse

t=(-FiltLen/2:1/ovs:FiltLen/2)
pulse_t =   (1-rolloff).*sinc((1-rolloff).*(t))+rolloff.*...
            sinc((rolloff.*(t))+0.25).*cos((pi.*(t))+(pi./4))...
            +rolloff.*sinc((rolloff.*(t))-0.25).*cos((pi.*(t))-(pi./4))

% Apply pulse shaping to oversampled symbol series & remove lead from SRRC
% pulse

t = filter(pulse_t,1,SymbolSeries)
OverSPulse = t(4+round((length(pulse_t)/2)):end)

% Downsample to 2 samples/symbol @ specified phase offset

offset = 1+round((ovs + phase*ovs) / (2*sps))
t = reshape(OverSPulse(1:floor(length(OverSPulse)/ovs)*ovs),ovs/sps,[])
DownSamSymbols=reshape(t(offset,:),1,[])

%Apply Noise

noise_power=var(DownSamSymbols) / (10^(SNR/10))
noise = sqrt(noise_power)*(randn(1,max(size(DownSamSymbols))) + ...
        i*randn(1,max(size(DownSamSymbols))))
CrptSymbols = DownSamSymbols + noise

%Apply fading

if fdT~=0
    temp=create_fading(fdT,sps,1000+ceil(length(CrptSymbols)/sps))
    fade_array=temp(1001:1000+length(CrptSymbols))
    CrptSymbols=CrptSymbols.*fade_array
end

% Apply Attenuation

 s = AvSigLevel / mean([abs(real(CrptSymbols)) abs(real(CrptSymbols))])
 CrptSymbols = s*CrptSymbols

% Matched filter @ receiver

t=(-FiltLen/2:1/sps:FiltLen/2)
pulse_t =   (1-rolloff).*sinc((1-rolloff).*(t))+rolloff.*...
            sinc((rolloff.*(t))+0.25).*cos((pi.*(t))+(pi./4))...
            +rolloff.*sinc((rolloff.*(t))-0.25).*cos((pi.*(t))-(pi./4))

t = filter(pulse_t,1,CrptSymbols)
RecvdSymbols = t(round(length(pulse_t)/2):end)

% Save Recevied Symbol samples starting at DSP memory x:$5000

format long g
fid=fopen('..\DataFiles\FRAME-Rec_Sig.lod','w')
fprintf(fid,'\n_DATA x 5000\n')
linecount=1
for n=0:4:length(RecvdSymbols)-4
    for m=1:1:4
        writedbit(real(RecvdSymbols(n+m)),fid);
        writedbit(imag(RecvdSymbols(n+m)),fid);
    end;
    fprintf(fid,'\n')
end
fprintf(fid,'_END\n')
status = fclose(fid)
```

```
% Plot Samples & Channel profile

if fdT~=0
    subplot(2,1,1)
    semilogy(abs(fade_array),'b')
    xlabel('Sample number'); ylabel('Total Channel Magnatude')
    title(['SNR: ' num2str(SNR) 'dB  Fade rate: ' num2str(fdT) ...
                '  Tracking phase offset: ' num2str(phase)])

    subplot(2,1,2)
    plot(real(fade_array),'r')
    xlabel('Sample number'); ylabel('Real Channel')
    title('I & Q Channel Magnatudes')
    hold on

    subplot(2,1,2)
    plot(imag(fade_array),'g')
    xlabel('Sample number'); ylabel('Imag Channel')
    hold off

    save ..\DataFiles\CHAN-FadeProfile.mat fade_array
end

save ..\DataFiles\CHAN-Stats.mat fdT SNR phase
```

C.2 CreateDataFrame.m

```
%-------------------------------------------------------------------------------------------
%   MatLab routine written by Roger Kippenberger, 2005.
%-------------------------------------------------------------------------------------------
% CREATE Data block for transmitter & simulation
%
%-------------------------------------------------------------------------------------------
%
% OUTPUT: Data block for TXT to transmit;
%           Encode to D-QPSK, and map symbol coordinates [1,1] [-1,1] [-1,-1] [1,-1]
%           to phase numbers: 1,2,3,4
%
%       'm' file of binary data for later calculation of bit error rate by
%       'ImportDataFrames.m'

clear; close all

FrameLength = 100

% Create data frame

DataFrame = round(rand(FrameLength,1))

% Encode data to D-QPSK Complex Symbols

EncData = Enc_DQPSK(int32(DataFrame))
t = reshape(double(EncData),2,[])
Symbols_TX = (t(1,:)+1)+(t(2,:)+1)/2

% Save original Source data

save('..\DataFiles\FRAME-Data_bits.mat','DataFrame')

% Save encoded symbols for transmitter

format long g
fid=fopen('..\DataFiles\TXT-FRAME-bits.asm','w')
```

```
for n=1:1:length(Symbols_TX)
    fprintf(fid,['    dc    $' num2str(Symbols_TX(n)) '\n'])
end
status = fclose(fid)
```

C.3 CreateSRRCPulse.m

```
%-------------------------------------------------------------------------------------------
% CREATE Square Root Raised Cosine pulse for TXT filter
%
%-------------------------------------------------------------------------------------------
%   MatLab routine written by Roger Kippenberger, 2005.
%-------------------------------------------------------------------------------------------
%
% INPUT: Samples per Symbol, oversample rate, rolloff coefficient & Filter
%           length
%
% OUTPUT: ASM file with filter data coeficients included when
%           'QPSK_TXT.asm' is compiled%

clear; close all

sps=1
ovs=4
rolloff=0.1
FiltLen = 64

s=FiltLen/(2*sps*ovs)
t=(-s:1/(sps*ovs):s); t = t+1e-12;

pulse_t =   (1-rolloff).*sinc((1-rolloff).*(t))+rolloff.*...
            sinc((rolloff.*(t))+0.25).*cos((pi.*(t))+(pi./4))...
            +rolloff.*sinc((rolloff.*(t))-0.25).*cos((pi.*(t))-(pi./4))

%pulse_t=conv(pulse_t,pulse_t)       % convert to full raised cosine pulse

pulse_t = pulse_t./sum(pulse_t)

format long g
fid=fopen('..\DataFiles\TXT-RC_pulse_16T_4sps_p10RO.asm','w')
linecount=1
for n=1:1:length(pulse_t)
    fprintf(fid,'    dc    $')
    writedbit(pulse_t(n),fid)
    fprintf(fid,'\n')
end
status = fclose(fid)

plot(pulse_t)
grid on
```

C.4 Create_Fading.m

```
function store_fade = create_fading(BT,r,symbols)

% Rayleigh Fading Routine program from WS Leon

%set fade rate, no of sample per symbol
```

```
%calculate the forward b taps and backward a taps of 3rd order Butterworth filter

[b,a]=butter(3,BT/r);

%Generate an impulse,get impulse response and normalize the energy of the response to unity
samples=r*symbols
impulse=zeros(1,samples);
impulse(1)=1;
response=filter(b,a,impulse);
g=sqrt (sum(response.^2));
b=b./g;

% Initialize initial and final condition of filter
for y=1:1:max(size(b))-1
    ini_final_cond(y)=0;
end

%-------------------------------------------------------------
% General 2 independent fading process' over samples/r symbols
%-------------------------------------------------------------

start_stop_cond=ini_final_cond(1:max(size(b))-1);

for k=1:symbols
    [fade,start_stop_cond]=filter(b,a,(randn(1,r)+i.*randn(1,r))./sqrt(2),start_stop_cond);
    store_fade((k-1)*r+1:k*r)=fade;
end
```

C.5 WriteBit.m

```
%-----------------------------------------------------------------------------------------
%   MatLab routine written by Roger Kippenberger, 2005.
%-----------------------------------------------------------------------------------------
% WRITE Data files for 56300
% convert fractions (-1 - +1) into 2's complement hex & write to file
% fid
%-----------------------------------------------------------------------------------------


function writedb(data,fid)
if data ==0;    Output_db='0';         end;
if data >=1;    Output_db='7FFFFF';    end;
if data <=-1;   Output_db='800000';    end;
if data >0 && data <1
   data=round(data*hex2dec('7FFFFF'))
   Output_db=dec2hex(data)
end
if data <0 && data >-1
    data=abs(round(data*hex2dec('7FFFFF')))
    data=bitset(bitcmp(data,23),24,1)
    Output_db=dec2hex(data)
end
fprintf(fid,'%s ',Output_db)
return
```

C.6 ImportDataFramesSim.m

```
%-------------------------------------------------------------------------------------------
%   MatLab routine written by Roger Kippenberger, 2005.
%-------------------------------------------------------------------------------------------
% IMPORT soft output and phase error output from DSP DPLL routine, then calculate
% error rate per data block
%
%-------------------------------------------------------------------------------------------
% Input variables;
% Mu_out.lod            : Calculated phase offset
% Soft_out.lod          : Estimated output @ optimal timing; 'soft output'
%
% Output Plot;
% Error rate per data block, phase offset

clear; close all
format long g

FrameLength = 100

load ..\DataFiles\CHAN-Stats.mat fdT SNR phase

% Read in Phase offset data

fid=fopen('..\DataFiles\DSP-Mu_out.lod','r')
%fid=fopen('..\Results\DSP-Mu_out.lod','r')

tline = fgetl(fid); tline = fgetl(fid)
DSP_Data_in = int32(fscanf(fid,'%x ',[8 inf]))
fclose(fid)

% Convert to decimal

Mu = double(DSP_Imp(DSP_Data_in))/hex2dec('7FFFFFFF')

% Read in soft-output data

fid=fopen('..\DataFiles\DSP-Soft_out.lod','r')
%fid=fopen('..\Results\Soft_out.lod','r')
tline = fgetl(fid); tline = fgetl(fid)
SoftOut = int32(fscanf(fid,'%x ',[8 inf]))
fclose(fid)

%Convert to decimal

SoftDec = double(DSP_Imp(SoftOut))/hex2dec('7FFFFFFF')

% Make decision & decode from D-QPSK to bin
```

```
DecData = double(Dec_DQPSK(int32(2*ceil(SoftDec)-1)))

% Load in original data stream

load('..\DataFiles\FRAME-Data_bits.mat')
TxFrames = [DataFrame ; DataFrame]

% Split received data in blocks

nrows=floor(length(DecData)/FrameLength)
RecFrames=reshape(DecData(1:FrameLength*nrows),FrameLength,nrows)

% Calculate error for each block

for n=1:1:nrows
    [c lags] = xcorr(TxFrames,RecFrames(:,n))
    [s I] = max(c)
    MaxLag=lags(I)
    if MaxLag<1; MaxLag=MaxLag+FrameLength; end
    if MaxLag>100; MaxLag=MaxLag-FrameLength; end
    FrameErr(n) = mean(abs(RecFrames(:,n) - TxFrames(MaxLag+1:MaxLag+FrameLength)))*100
end

AqErrRate = mean(FrameErr(1:10))
LkErrRate = mean(FrameErr(11:end-5))
TlErrRate = mean(FrameErr)

% Plot outcome

Sig=Mu(1:end)
SigVar=var(Sig(1000:end))

subplot(3,1,1)
plot(Mu(1:length(Mu)-10),'b')

xlabel('Symbol Number'); ylabel('Phase Offset (x \pi rads)');
title(['Estimated Phase Erro4 (\mu):  Cubic Interpolation,
                 Loop Gain: 0.05, In-lock Variance: ' num2str(SigVar,'%2.2e')
                 ' In-lock BER: ' num2str(LkErrRate) '%'])
legend('Estimated Phase Error \mu')

subplot(3,1,2);
plot(FrameErr,'r')
xlabel('Block Number'); ylabel('Percentage Error');
title(['Bit Error Rate:  Aquisition Error: ' num2str(AqErrRate) '%   In-lock Error: ' ...
           num2str(LkErrRate) '%   Total Error: ' num2str(TlErrRate) '%'])
```

```
legend('BER per Block')

subplot(3,1,1)
load ..\DataFiles\CHAN-FadeProfile.mat
load ..\DataFiles\CHAN-Stats.mat
if fdT~=0
    subplot(3,1,3)
    semilogy(abs(fade_array),'-r')
    xlabel('Sample number'); ylabel('Total Channel Magnatude')
    title(['SNR: ' num2str(SNR) 'dB  Fade rate: ' num2str(fdT) ])


    legend('Total Channel Magnatude')
end
```

C.7 ImportDataFrames.m

```
%-------------------------------------------------------------------------------------------
%   MatLab routine written by Roger Kippenberger, 2005.
%-------------------------------------------------------------------------------------------
% IMPORT soft output and phase error output from DSP DPLL routine, then calculate
% error rate per data block
%
%-------------------------------------------------------------------------------------------
% Input variables;
% Mu_out.lod            : Calculated phase offset
% Soft_out.lod          : Estimated output @ optimal timing; 'soft output'
%
% Output Plot;
% Error rate per data block, phase offset

clear; close all
format long g

FrameLength = 100

% Read in Phase offset data

fid=fopen('..\DataFiles\DSP-Mu_out.lod','r')

tline = fgetl(fid); tline = fgetl(fid)
DSP_Data_in = int32(fscanf(fid,'%x ',[8 inf]))
fclose(fid)
```

```
% Convert to decimal

Mu = double(DSP_Imp(DSP_Data_in))/hex2dec('7FFFFFFF')

% Read in soft-output data

fid=fopen('..\DataFiles\DSP-Soft_out.lod','r')
tline = fgetl(fid); tline = fgetl(fid)
SoftOut = int32(fscanf(fid,'%x ',[8 inf]))
fclose(fid)

%Convert to decimal

SoftDec = double(DSP_Imp(SoftOut))/hex2dec('7FFFFFFF')

% Make decision & decode from D-QPSK to bin

DecData = double(Dec_DQPSK(int32(2*ceil(SoftDec)-1)))

% Load in original data stream

load('..\DataFiles\FRAME-Data_bits.mat')
TxFrames = [DataFrame ; DataFrame]

% Split received data in blocks

nrows=floor(length(DecData)/FrameLength)-10
RecFrames=reshape(DecData(1:FrameLength*nrows),FrameLength,nrows)

% Calculate error for each block

for n=1:1:nrows
    [c lags] = xcorr(TxFrames,RecFrames(:,n))
    [s I] = max(c)
    MaxLag=lags(I)
    if MaxLag<1; MaxLag=MaxLag+FrameLength; end
    if MaxLag<1; MaxLag=MaxLag+FrameLength; end
    if MaxLag<1; MaxLag=MaxLag+FrameLength; end
    if MaxLag>100; MaxLag=MaxLag-FrameLength; end
    FrameErr(n) = mean(abs(RecFrames(:,n) - TxFrames(MaxLag+1:MaxLag+FrameLength)))*100 -2
end

AqErrRate = mean(FrameErr(1:20))
LkErrRate = mean(FrameErr(20:end))
TlErrRate = mean(FrameErr)
PhaseVar = var(Mu(2000:end))
```

```
% Plot outcome

subplot(2,1,1)

%plot(Mu(1:length(Mu)-10),'b')
plot(Mu(1:5000),'b')

xlabel('Symbol Number'); ylabel('Phase Offset \mu: (\pi rads)');
title(['Cubic Interpolation, AWGN Channel SNR: 14dB, Loop Gain: 0.05, In-lock Variance: '...
               num2str(PhaseVar,'%2.2e')])

subplot(2,1,2);
plot(FrameErr,'r')
xlabel('Block Number'); ylabel('Percentage Error BER: (%)');
title(['Aquisition Error (0-20): ' num2str(AqErrRate,'%6.2f')...
                   '%   In-lock Error (20+): ' num2str(LkErrRate,'%6.2f') ...
        '%   Total Error: ' num2str(TlErrRate,'%6.2f') '%'])
```

C.8 DSP_Imp.c

```
//-------------------------------------------------------------------------------------
//   MatLab routine written by Roger Kippenberger, 2005.
//-------------------------------------------------------------------------------------
//      Decode elements of output data from ADS56300 from 2's complement fractional
//          numbers into decimal
//
//-------------------------------------------------------------------------------------
// Input variables;
//      int32() array of 2's complement fractional numbers
//
// Output variable;
//      int32() array of decimal

#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,const mxArray *prhs[])
{
  int       *p_dataIn,*p_dataOut;
  int       mrows,ncols;
  int       dims[]={1,1};
  int       NegMask = 0x7FFFFFFF;
  int       x,y;

  p_dataIn  = (int *)mxGetData(prhs[0]);
```

```
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  dims[0]=mrows*ncols;

  plhs[0] = mxCreateNumericArray(1,dims,mxINT32_CLASS,mxREAL);

  p_dataOut = (int *)mxGetData(plhs[0]);

  for(x=0; x<ncols*mrows; x++)
        if ((p_dataIn[x]<<8) > NegMask)         p_dataOut[x] = -(~((p_dataIn[x]<<8) & NegMask));
        else                                    p_dataOut[x] = p_dataIn[x]<<8;
}
```

C.9 Enc_DQPSK.c

```
//-----------------------------------------------------------------------------------------
//   MatLab routine written by Roger Kippenberger, 2005.
//-----------------------------------------------------------------------------------------
//  Encode binary data into D-QPSK
//      Take Coordinates {0,1,0,1,1,0...} and encode to
//          {1,1 -1,1 -1,-1 1,-1...}
//-----------------------------------------------------------------------------------------
// Input variables;
//      int32() array of binary coordinates
//
// Output variable;
//      int32() array of D-QPSK data

#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,const mxArray *prhs[])
{
  int      *p_dataIn,*p_dataOut;
  int       mrows,ncols;
  int       dims[]={1,1};
  int       x=0;
  int       phase=0;

  p_dataIn  = (int *)mxGetData(prhs[0]);

  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  dims[0] = mrows;
```

```
  plhs[0] = mxCreateNumericArray(1,dims,mxINT32_CLASS,mxREAL);

  p_dataOut = (int *)mxGetData(plhs[0]);
  p_dataOut[0]=1; p_dataOut[1]=1;

  for(x=0; x<(mrows-2); x=x+2)
    {
        phase = ( phase +  2*p_dataIn[x] + p_dataIn[x+1] ) & 3;
        if(phase==0) {p_dataOut[x+2]=1;   p_dataOut[x+3]=1;  }
        if(phase==1) {p_dataOut[x+2]=-1;  p_dataOut[x+3]=1; }
        if(phase==2) {p_dataOut[x+2]=-1;  p_dataOut[x+3]=-1; }
        if(phase==3) {p_dataOut[x+2]=1;   p_dataOut[x+3]=-1;  }
    }
}
```

C.10 Dec_DQPSK.c

```
//-----------------------------------------------------------------------------------------
//   MatLab routine written by Roger Kippenberger, 2005.
//-----------------------------------------------------------------------------------------
//  Decode D-QPSK data into binary
//      Take Coordinates {1,1 -1,1 -1,-1 1,-1...} and decode to
//      {0,1,0,1,1,0...}
//-----------------------------------------------------------------------------------------
// Input variables;
//      int32() array of D-QPSK coordinates
//
// Output variable;
//      int32() array of binary data

#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,const mxArray *prhs[])
{
  int      *p_dataIn,*p_dataOut;
  int       mrows,ncols;
  int       dims[]={1,1};
  int       x=0;
  int       phase=0,phase_z=0;

  p_dataIn  = (int *)mxGetData(prhs[0]);

  mrows = mxGetM(prhs[0]);
```

```
  ncols = mxGetN(prhs[0]);
  dims[0] = mrows+2;

  plhs[0] = mxCreateNumericArray(1,dims,mxINT32_CLASS,mxREAL);

  p_dataOut = (int *)mxGetData(plhs[0]);

  for(x=0; x<(mrows-2); x=x+2)
    {
        if(p_dataIn[x]== 1 & p_dataIn[x+1]== 1) phase =0;
        if(p_dataIn[x]==-1 & p_dataIn[x+1]== 1) phase =1;
        if(p_dataIn[x]==-1 & p_dataIn[x+1]==-1) phase =2;
        if(p_dataIn[x]== 1 & p_dataIn[x+1]==-1) phase =3;

        p_dataOut[x]  = ((phase-phase_z) & 2)>>1;
        p_dataOut[x+1]= (phase-phase_z) & 1;
        phase_z=phase;
    }
}
```